

Grado Universitario en Ingeniería Informática

2018-2019

Trabajo Fin de Grado

“Análisis y aplicación de técnicas de
aprendizaje automático para clasificación
de reseñas en redes sociales”

Óscar Martín de la Fuente Sanz

Tutor

Israel González Carrasco

Leganés, marzo de 2019



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

Determinar el sentimiento o la polaridad de un texto ha sido una de las tareas con mayor investigación dentro del ámbito del procesamiento del lenguaje. Aunque inicialmente se tomaban enfoques que analizaban la sintaxis y la semántica del texto, esto evolucionó hacia el empleo de técnicas de aprendizaje automático para realizar la clasificación del texto. En los últimos años, el crecimiento del campo del aprendizaje profundo ha provocado grandes mejoras respecto a los métodos tradicionales de análisis de sentimiento.

En este proyecto se pretende analizar los distintos métodos de aprendizaje automático existentes actualmente para la clasificación de texto, desde las técnicas que sirven como base hasta los modelos más recientes, y aplicar los mismos a un conjunto de datos de una red social de valoración de servicios, con el propósito de encontrar el mejor modelo posible que permita conocer la polaridad de una reseña.

Resulta interesante estudiar los posibles modelos, pues hoy en día, con la completa implantación de Internet en la sociedad, son muchos los usuarios que aprovechan las redes sociales para dar su opinión sobre los productos o servicios que utilizan, y por ello, es realmente útil para los negocios poder conocer lo que piensan sus usuarios de manera automática.

Palabras clave: Aprendizaje automático; Análisis de sentimiento; Redes sociales; Inteligencia Artificial; Word embeddings.

AGRADECIMIENTOS

Me gustaría expresar mi más profundo agradecimiento a todos los que me han acompañado y ayudado tanto en la elaboración de este trabajo como a lo largo de toda mi vida académica.

En primer lugar, a mi tutor Israel, por su apoyo y guía durante toda la realización del proyecto, así como por su paciencia a la hora de prestarme tiempo y ayuda.

A mis padres, por todo. Porque todo lo que soy y lo que he hecho se lo debo a ellos, que siempre han hecho cualquier cosa por mí.

A todos mis compañeros de clase en esta etapa, por las risas y las infinitas horas de prácticas. Y de entre ellos, sobre todo a Juanjo, Pozo y Saúl, sin ellos esto no habría sido igual.

Al resto de mis familiares y amigos, por siempre preocuparse por mí y darme cariño.

También a cualquier profesor cuyas enseñanzas hayan aportado algún conocimiento que me haya ayudado a llegar hasta aquí.

Por último, pero quizá más importante, a Jenny, porque si puedo alcanzar mis metas es gracias a ella, por estar en los buenos y en los malos momentos.

*“I visualize a time when we will be to robots what dogs are to humans, and I’m
rooting for the machines”*
— Claude Shannon

ÍNDICE GENERAL

1. INTRODUCCIÓN	1
1.1. Motivación del trabajo	1
1.2. Objetivos	2
1.3. Estructura del documento	3
2. ESTADO DE LA CUESTIÓN	4
2.1. Análisis de sentimiento	4
2.2. Aprendizaje automático para clasificación de sentimiento	8
2.2.1. Algoritmos de aprendizaje supervisado	13
2.2.2. El modelo bag-of-words	26
2.3. Word embeddings	31
2.3.1. Word2Vec	35
2.3.2. Transferencia de aprendizaje y aprendizaje profundo	39
2.4. La evolución de los word embeddings	47
2.5. Clasificación multiclase	53
3. ANÁLISIS Y DESARROLLO DEL PROBLEMA	56
3.1. Planteamiento	56
3.2. Metodología	60
3.3. El conjunto de datos	63
3.3.1. Obtención de los datos	69
3.3.2. Preprocesamiento de los datos	72
3.4. Herramientas y tecnologías	76
3.4.1. Especificaciones del entorno	76
3.4.2. Tecnologías	77
4. EXPERIMENTACIÓN	81
4.1. Experimentación previa	82
4.2. Experimentación con el modelo bag-of-words	85
4.3. Experimentación con word embeddings	90
4.4. Experimentación con ULMFiT	96

5. EVALUACIÓN DE RESULTADOS	98
5.1. Métricas de evaluación	98
5.2. Evaluación del modelo bag-of-words.	100
5.2.1. Bayes ingenuo	100
5.2.2. SVM	105
5.2.3. Random Forest	110
5.2.4. Regresión logística	113
5.2.5. Comparativa general de bag-of-words.	118
5.3. Evaluación de los modelos con word embeddings	120
5.3.1. Red neuronal convolucional profunda en forma piramidal.	121
5.3.2. LSTM bidireccional con reducción en dos dimensiones	128
5.4. Evaluación de ULMFiT	132
5.5. Discusión y comparativa final de modelos	135
6. MARCO REGULADOR	140
6.1. Legislación aplicable	140
6.2. Estándares técnicos y propiedad intelectual	142
7. ENTORNO SOCIO-ECONÓMICO.	145
7.1. Planificación del proyecto	145
7.2. Presupuesto del proyecto	149
7.2.1. Coste de personal.	149
7.2.2. Coste de material.	149
7.2.3. Costes adicionales	150
7.2.4. Coste final	150
7.3. Impacto.	151
8. CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO	153
BIBLIOGRAFÍA.	155

ÍNDICE DE FIGURAS

1.1	Logo de Yelp	1
2.1	Número de publicaciones por año en Scopus	5
2.2	Funcionamiento general de un problema de <i>machine learning</i>	9
2.3	Representación de una clasificación binaria con SVM	18
2.4	Máximización de márgenes en una clasificación binaria con SVM	19
2.5	Representación de una clasificación no lineal con SVM	20
2.6	Transformación mediante kernel a un problema no lineal	20
2.7	Un clasificador <i>ensemble</i> de redes neuronales	23
2.8	Un árbol de decisión para clasificación	25
2.9	Estructura de una red neuronal artificial	32
2.10	Esquema del modelo de una red neuronal artificial	33
2.11	Continuous Bag of Words simple	36
2.12	Continuous Bag of Words con múltiples contextos	37
2.13	Funcionamiento de Skip-gram	38
2.14	Arquitectura de una red neuronal convolucional	42
2.15	Red de gran memoria a corto plazo	44
2.16	ULMFiT	49
2.17	Modelo BERT	52
3.1	Metodología de trabajo	61
3.2	Ejemplo de reseña en Yelp	64
3.3	Estadísticas descriptivas de las valoraciones	65
3.4	Distribución de las reseñas	66
3.5	Longitud de las reseñas en función de sus estrellas	67
3.6	Diagrama de cajas de longitud de la reseña en función de las estrellas	69
3.7	Muestra del conjunto de datos	70
3.8	Reseña no procesada	73
3.9	Reseña procesada	76

4.1	Red neuronal convolucional profunda en forma piramidal	94
4.2	LSTM bidireccional con reducción en dos dimensiones	95
4.3	Arquitectura AWD-LSTM	97
5.1	Matriz de confusión de Naïve Bayes binario	103
5.2	Matriz de confusión de Naïve Bayes multiclase	105
5.3	Matriz de confusión de SVM binario	108
5.4	Matriz de confusión de SVM multiclase	110
5.5	Matriz de confusión de Random Forest multiclase	113
5.6	Matriz de confusión de la regresión logística binaria	115
5.7	Matriz de confusión de la regresión logística multiclase	118
5.8	Comparativa de resultados de bag-of-words	119
5.9	Matriz de confusión de DPCNN en el conjunto binario	125
5.10	Matriz de confusión de DPCNN en el conjunto multiclase	128
5.11	Matriz de confusión de BLSTM en el conjunto binario	130
5.12	Matriz de confusión de BLSTM en el conjunto multiclase	132
5.13	Matriz de confusión de ULMFiT en el conjunto binario	134
5.14	Matriz de confusión de ULMFiT en el conjunto multiclase	135
5.15	Comparativa de modelos	136
5.16	Comparativa de modelos con resultados originales	138
6.1	Planificación de la gestión de la IA en Europa	141
7.1	Diagrama de Gantt de planificación	148

ÍNDICE DE TABLAS

2.1	Comparativa de modelos previos según el tipo de aprendizaje	12
2.2	Comparativa de modelos profundos con word embeddings	46
2.3	Comparativa de resultados en clasificación multiclase	54
3.1	Campos de datos de review	65
3.2	Media de longitud de la reseña en función de las estrellas	68
3.3	Especificaciones del entorno de desarrollo	76
4.1	Experimentación previa con conjunto stemmatizado	84
4.2	Experimentación previa con el doble de datos	84
4.3	Parámetros de experimentación con Naïve Bayes	87
4.4	Parámetros de experimentación con SVM	88
4.5	Parámetros de experimentación con Random Forest	89
4.6	Parámetros de experimentación con Regresión logística	90
5.1	Matriz de confusión	99
5.2	Resultados de bayes ingenuo en el conjunto binario	101
5.3	Resultados de bayes ingenuo en el conjunto multiclase	103
5.4	Resultados de SVM en el conjunto binario	105
5.5	Segunda experimentación con SVM en el conjunto binario	107
5.6	Resultados de SVM en el conjunto multiclase	108
5.7	Resultados de Random Forest en el conjunto binario	110
5.8	Resultados de Random Forest en el conjunto multiclase	112
5.9	Resultados de la regresión logística en el conjunto binario	113
5.10	Resultados de la regresión logística en el conjunto multiclase	116
5.11	Resultados de bag-of-words en conjunto binario	119
5.12	Resultados de bag-of-words en conjunto multiclase	119
5.13	Resultados de DPCNN en el conjunto binario	122
5.14	Resultados de DPCNN en el conjunto multiclase	126
5.15	Resultados de BLSTM en el conjunto binario	129

5.16	Resultados de BLSTM en el conjunto multiclase	131
5.17	Mejor resultado de ULMFiT en el conjunto binario	133
5.18	Mejor resultado de ULMFiT en el conjunto multiclase	134
5.19	Resultados finales en el conjunto binario	136
5.20	Resultados finales en el conjunto multiclase	136
6.1	Resumen de licencias de uso de las tecnologías empleadas	144
7.1	Coste de personal	149
7.2	Coste de material	150
7.3	Costes adicionales	150
7.4	Coste final	151

1. INTRODUCCIÓN

1.1. Motivación del trabajo

La importancia de las redes sociales y el impacto que tienen en muchos de los aspectos de la sociedad es indiscutible. De acuerdo a estadísticas recientes (Kemp, 2018), cada mes más de 3 mil millones de personas utilizan las redes sociales, lo que constituye más de un 42% de la población mundial. Teniendo en cuenta este dato, desde el punto de vista de la investigación, las redes sociales representan una fuente de datos muy útil para realizar estudios, tanto por la gran cantidad de datos disponibles como por las distintas posibilidades que ofrecen. Del mismo modo, para el sector empresarial se materializa en una oportunidad de extraer información valiosa sobre sus usuarios y sus productos a través de la aplicación de, por ejemplo, técnicas de análisis de datos o de análisis predictivo con aprendizaje automático.

En este contexto, se conoció que la red social Yelp, un sitio web de búsqueda y reseña de servicios muy usado en América y también en Europa en menos grado, proporcionaba de forma abierta un subconjunto de los datos de su aplicación para su uso con fines académicos o de investigación. Este subconjunto incluye desde información sobre los servicios, los usuarios y las reseñas hasta las imágenes de las mismas, lo que permite la utilización de técnicas como minería de datos, minería de grafos o reconocimiento de imágenes a través de aprendizaje profundo, entre otras.



Fig. 1.1. Logo de Yelp

La clasificación o el análisis de datos provenientes de redes sociales ha sido un tema muy utilizado para trabajos de este tipo, no obstante, los últimos años han signifi-

cado grandes avances para el análisis de sentimiento o, en términos de aprendizaje automático, la clasificación de texto, con publicaciones como la de Peters et al. (2018) o la de Howard y Ruder (2018), que han logrado encontrar modelos con una precisión muy alta. Por ello, tras estudiar las distintas posibilidades que ofrecía el conjunto de datos de Yelp, se encontró que resultaba muy interesante analizar sobre éste los distintos modelos de aprendizaje existentes para la clasificación de texto, desde los modelos que han pasado a servir como base hasta los modelos más recientes e innovadores.

1.2. Objetivos

El objetivo de este trabajo es estudiar los numerosos métodos existentes para el análisis de sentimiento a través de aprendizaje automático, experimentando con ellos sobre el conjunto de datos de la red social Yelp con el fin de encontrar el mejor modelo posible para la clasificación de reseñas. De esta forma, se pretende lograr un modelo que sea capaz de predecir el número de estrellas que tiene una reseña a partir del texto de la misma. Por supuesto, dada la dimensión tanto temporal como espacial de un Trabajo Fin de Grado, no se busca probar todo tipo de algoritmos, modelos y tecnologías, sino que se explorará lo más relevante de acuerdo a la literatura consultada (que se especificará en el capítulo de *Estado de la cuestión*).

Adicionalmente, existen otros objetivos subyacentes a este objetivo principal:

- Comprender y conocer los distintos modelos de aprendizaje automático existentes para la clasificación de texto, así como las distintas tecnologías que se utilizan para implementar estos modelos.
- Lograr resultados cercanos a los que se han conseguido en investigaciones del mismo área.
- Contribuir a este campo como un estudio general de los modelos posibles para el análisis de sentimiento.
- Desarrollar un proyecto de investigación en el área de la inteligencia artificial, entender y aplicar una metodología apropiada a dicho proyecto y alcanzar un

resultado final.

1.3. Estructura del documento

Este documento sigue una estructura definida que se detalla a continuación y que comienza con este primer apartado de introducción.

Primero, se realizará una revisión de la literatura especializada para comprender los fundamentos teóricos y prácticos de los métodos existentes para estudiar las distintas soluciones al problema.

Posteriormente, se describirá la metodología a seguir, las herramientas a utilizar, y, constituyendo el núcleo práctico del trabajo, el desarrollo de la experimentación en busca de un resultado final.

Por último, antes de las conclusiones finales, en las cuáles también se detallarán las futuras líneas de trabajo a partir de este trabajo, se realizará un análisis sobre el marco regulador relativo a esta investigación en el capítulo 6 y se efectuará un análisis del impacto socio-económico de este estudio, incluyendo la planificación y el presupuesto del mismo, en el capítulo 7.

Al final del documento se podrá encontrar la bibliografía del trabajo, en la cuál aparecerán referenciadas todas las fuentes consultadas durante su realización. Después de esta sección, y no formando parte de lo que sería el cuerpo principal del trabajo, se encontrarán los anexos.

2. ESTADO DE LA CUESTIÓN

2.1. Análisis de sentimiento

El análisis de sentimiento, también conocido como minería de opinión, es una rama del campo del Procesamiento del Lenguaje Natural que trata de extraer información subjetiva a partir de un texto y es uno de los campos con mayor crecimiento de las ciencias de la computación. Generalmente, se trata del uso de una serie de técnicas, métodos y herramientas de análisis de texto con el fin de conocer la polaridad de una opinión, es decir, si la opinión es negativa o positiva, o en algunos casos, si ésta es negativa, neutra o positiva (Dave, Lawrence & Pennock, 2003). No obstante, conforme ha avanzado este área se han producido enfoques que han profundizado más aún, dejando a un lado la búsqueda de una polaridad y centrándose en la extracción de un conjunto de emociones a partir del texto.

El interés por el estudio de técnicas para extraer la opinión de un texto existe desde que la sociedad ha tenido la capacidad de publicar sus propias opiniones (por ejemplo, a través del periódico), y tiene una razón de ser muy lógica, y es que cuando se piensa en comprar un producto o tomar alguna decisión, se recurre a conocer la opinión de amigos o familiares. También, por parte de organizaciones para conocer la opinión pública o de las compañías para estar al tanto de las tendencias.

Esto son ejemplos básicos que buscan explicar el origen de los esfuerzos por analizar la opinión de un texto, pero a día de hoy el foco en este tema puede estar puesto desde muchas perspectivas. En el contexto de los sitios de reseñas, la utilización de estas técnicas podría servir también para corregir errores de los usuarios, cuando la valoración de una reseña no concuerda con el texto. También puede resultar muy útil en sistemas de recomendación, en los que la automatización del análisis podría ayudar a no recomendar contenido muy negativo.

Pero quizá el sector que más se puede aprovechar o que más información puede extraer a través de esta técnica es el de la inteligencia, es decir, el de la *business*

*intelligence*¹, pues puede permitir a las empresas conocer las opiniones sobre sus productos y puede saber a través de estas opiniones el motivo por el que los usuarios compran, o dejan de comprar, sus productos. Esto se podría conocer a simple vista, leyendo directamente las opiniones, pero un sistema que las analizase podría extraer una generalización de las opiniones de forma automática.

Estos son algunos de los intereses que han impulsado la evolución de este área (más han sido estudiados por Pang y Lee (2008)), pero hasta hace unos años no estaba lo suficientemente desarrollada como para plantearse estas soluciones.

Inicialmente, para esta tarea se empleaban diversas técnicas del área del procesamiento del lenguaje, como el análisis sintáctico, el análisis semántico y el análisis morfológico. No obstante, la verdadera explosión de este campo no llegó hasta principios del siglo XXI, con la expansión de Internet y las redes sociales. De acuerdo a Liu (2010), la falta de investigación del análisis de sentimiento previa a este hecho se debe a que no existían grandes cantidades de texto opinado para analizar, pero la Web cambió completamente esta situación, gracias a la aparición de foros y sitios en los que los usuarios pudieran expresar libremente su opinión.

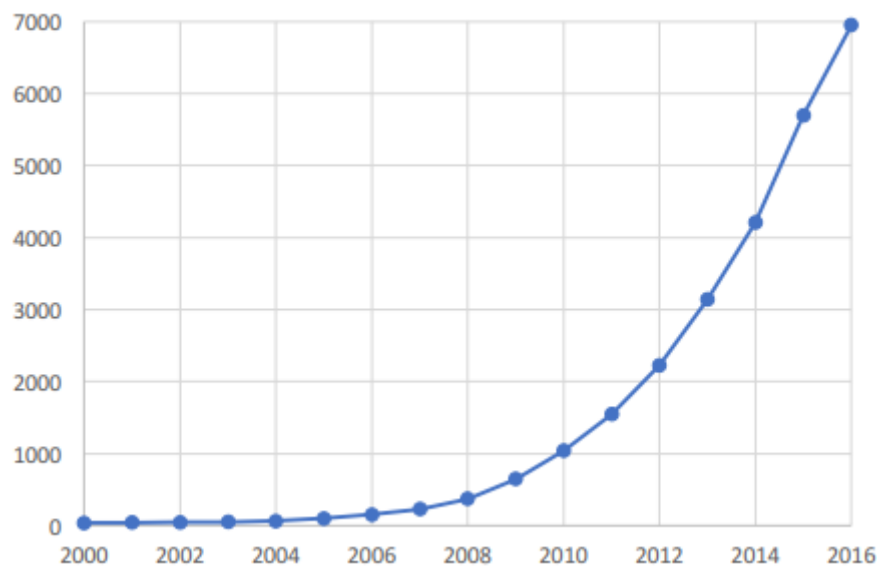


Fig. 2.1. Número de publicaciones por año en Scopus (Mäntylä et al., 2016)

¹inteligencia empresarial

En la Figura 2.1 se puede apreciar el gran crecimiento que ha tenido el área del análisis de sentimiento en los últimos años, tomando como medida el número de publicaciones de investigación relativas a este tema. Es en este momento en el que, una vez que existen una gran variedad de fuentes de datos que analizar, se comienza a aplicar métodos computacionales para esta tarea, como técnicas de *machine learning*². Así, se llega a los dos tipos de enfoques que existen actualmente: enfoques basados en el léxico y el aprendizaje automático.

Una de las publicaciones que más relevancia ha tenido en el área y que fue una de las impulsoras de esta tendencia creciente fue la de Pang, Lee y Vaithyanathan (2002), que estableció los fundamentos que aún en los estudios más recientes se toman como base y que será una de las referencias principales a lo largo de todo este trabajo. En este trabajo toman como conjunto de datos una serie de reseñas sobre películas y se plantean clasificarlas en positivas o negativas, lo cual significó un paso adelante respecto al estado del área en aquel entonces, cuándo los investigadores se centraban en técnicas para averiguar el tema del texto. Juntando el hecho de que se exploraban nuevas formas de clasificar el sentimiento, a través de aprendizaje automático y no de las técnicas tradicionales de análisis del lenguaje, al hecho de centrarse en la polaridad del texto, quedaron marcadas las líneas a seguir en futuros estudios.

Es necesario recalcar que las técnicas empleadas en muchos de los modelos recientes y que han sido producto de este crecimiento no son nuevas, pues fueron definidas décadas atrás, pero no se contaba entonces con la disponibilidad suficiente de datos sobre los que aplicarlas. Por ejemplo, el modelo de *bag-of-words*, que toman como base Pang, Lee y Vaithyanathan en su trabajo, que aquí se estudiará posteriormente y que sigue siendo un modelo con buenos resultados a día de hoy, había sido ya mencionado a mediados del siglo anterior por Harris (1954).

Otro aspecto a tener en cuenta es el desarrollo de los algoritmos de aprendizaje profundo, que se convirtió en una realidad alrededor de 2010 y, como se verá más tarde en este documento, ha tenido también un gran impacto en este campo. Si volvemos a fijar la vista en la Figura 2.1, este es uno de los momentos en los cuáles la gráfica toma una pendiente más creciente.

²aprendizaje automático

A partir de estos puntos clave, la mayoría de las aproximaciones destacadas dentro del campo han estado ligadas a la utilización de técnicas de aprendizaje automático. En el siguiente apartado se tratará este aspecto, pero a continuación se presta atención de forma más práctica al análisis de sentimiento.

A priori, la tarea de análisis de sentimiento de una frase puede parecer muy simple. Si se toma como ejemplo la frase “*La comida de este restaurante es muy buena*”, a simple vista se entiende que *buena* indica que la frase es *positiva*. Es más, si se tuvieran en cuenta más polaridades que *negativa* o *positiva*, un analizador ideal detectaría que *muy* indica un grado superlativo y clasificaría la frase como *muy positiva*. Lo mismo debería ocurrir si la frase fuera “*La comida de este restaurante es buenísima*”. Sin embargo, y como muy bien detalla Liu (2010) en su publicación, la problemática es más compleja.

Uno de estos problemas sería la explicitud de la frase. Se toman por ejemplo dos oraciones:

1. *La comida de este restaurante es buena*
2. *Volvería a comer a este restaurante*

La frase 1 representa una opinión **explícita**, si se analizase como en el párrafo anterior se vería que es una frase positiva, es decir, sin ningún significado oculto la polaridad de la frase queda clara. El problema aparece con la frase 2, que contiene una opinión **implícita**, es decir, en la frase no se ve ninguna palabra que indique positividad, pero la connotación de la misma es claramente positiva, pues quiere decir que quiere volver al restaurante porque la comida ha sido buena.

Otro problema de esta tarea surge cuando se incluyen palabras cuyo significado por sí solas es contrario al de la opinión en su conjunto. Ilustremos este problema con otro ejemplo:

Llegué muy mal, pero en el restaurante se portaron conmigo

Desde un punto de vista humano, se entiende que la frase tiene un sentido positivo. No obstante, en un análisis automático de la frase, podría ocurrir que se tuviese en cuenta que la palabra *mal* es negativa, y que no hay ninguna otra palabra positiva,

con lo cuál se podría clasificar la frase como una opinión negativa. Esta ambigüedad también se da cuando en la opinión se utiliza un tono de ironía o sarcástico.

La utilización de técnicas de aprendizaje automático facilita la solución de algunos de estos problemas. Un clasificador automático no tiene en cuenta de ninguna manera el significado o la connotación de una palabra dentro de una frase, pero si está bien entrenado puede detectar las palabras que indican una polaridad u otra. La capacidad de generalización de los clasificadores logra aventajar a las técnicas de análisis del lenguaje natural (como probaron Pang, Lee y Vaithyanathan) gracias al entrenamiento, pero al no tener en cuenta ningún tipo de semántica podría tener dificultades cuando la opinión que se intenta clasificar es implícita o está escrita de forma ambigua.

Se estudiarán distintos modelos de clasificación que logren la mayor exactitud posible y que solucionen estos problemas.

2.2. Aprendizaje automático para clasificación de sentimiento

Si se mira desde el punto de vista del Aprendizaje automático, que es el enfoque que se le da en este trabajo, el problema del análisis de sentimiento es un problema de clasificación, en el cuál la entrada al algoritmo de aprendizaje sería el texto a clasificar. Por tanto, es indispensable comprender tanto teórica como prácticamente los distintos algoritmos, así como la forma en la que se aplican para el análisis de sentimiento.

El aprendizaje automático es un campo de la Inteligencia Artificial, y como tal, está muy fundamentado en los modelos matemáticos y estadísticos planteados desde principios del siglo XX. Al ser una rama que ha sido muy estudiada, podrían encontrarse diversas definiciones. Una muy correcta podría ser la de Domingos (2012), que dice que “el aprendizaje automático es un conjunto de algoritmos capaces de aprender a realizar ciertas tareas a partir de la generalización de ejemplos”.

A día de hoy, está tan desarrollado que es utilizado en muchos y muy diferentes campos: análisis de mercado, sistemas de recomendaciones, posicionamiento de pu-

blicidad, detección de fraudes y sistemas de predicciones de todo tipo, entre otros más. Y aún así, es un ámbito que está en constante evolución, por lo que continuamente surgen nuevas aplicaciones.

Como ya se mencionó en el anterior apartado, uno de los primeros trabajos en los que se planteó la utilización de *machine learning* para clasificación de sentimiento fue en Pang et al. (2002), en el cuál se planteaba la utilización de algoritmos de aprendizaje supervisado. ¿Qué es el aprendizaje supervisado? Para entenderlo se debe comprender el funcionamiento general del modelado de un problema de aprendizaje automático:

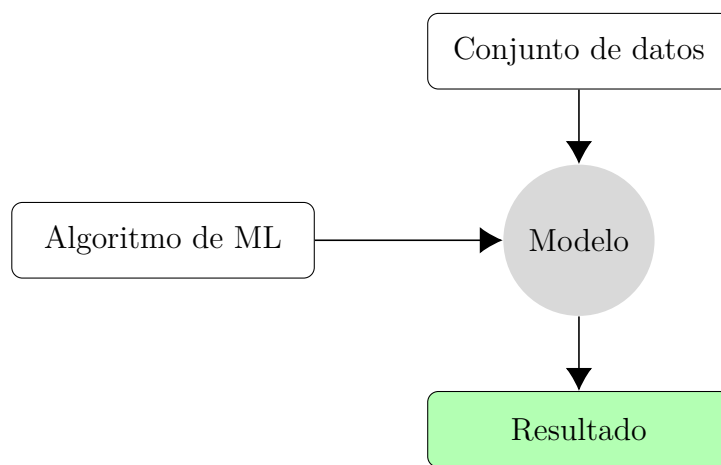


Fig. 2.2. Funcionamiento general de un problema de *machine learning*

El esquema de la figura 2.2 sería una abstracción general de las dos principales fases de un problema de aprendizaje automático:

1. **Entrenamiento:** un conjunto de ejemplos de entradas con sus correspondientes resultados/etiquetas se le pasa a un algoritmo, que construye lo que se llama un “modelo” aplicando el algoritmo sobre los datos de ejemplo, de forma que el algoritmo aprenda qué entradas producen qué resultados. Una vez el modelo ha “aprendido” de los datos de ejemplo, el resultado es el modelo entrenado capaz de actuar sobre otros datos distintos a los de ejemplo y predecir sus salidas.
2. **Test:** un conjunto de ejemplos sin los resultados se le pasa a un modelo ya entrenado, que da como resultado una salida que debería ser el resultado es-

perado, dependiendo de lo bien entrenado que esté el modelo.

Posteriormente, cuando se explique cada uno de los algoritmos, se entrará más en detalle en cada una de estas fases.

Teniendo en mente el esquema básico, se puede explicar los diferentes tipos de aprendizaje:

- **Aprendizaje supervisado:** es el tipo de aprendizaje más común y su funcionamiento se puede extraer de su nombre, el proceso de entrenamiento está “supervisado”, pues el algoritmo cuenta tanto con los valores de entrada como con sus correspondientes salidas. De este modo, el algoritmo realiza predicciones sobre los datos de entrenamiento y la supervisión consiste en corregir al algoritmo si es equivoca para que no se equivoque en el futuro.

Los problemas de aprendizaje supervisado se pueden dividir en dos grupos:

- **Clasificación:** es un problema en el cuál los datos de entrada deben clasificarse en determinadas categorías. Por ejemplo, el caso particular de este trabajo sería un problema de clasificación binaria, pues se deben clasificar las opiniones en determinadas categorías.
- **Regresión:** es un problema en el cuál la salida es un número real, como por ejemplo un peso o un precio.

Se podría representar el aprendizaje como una función

$$Y = f(X) \tag{2.1}$$

donde X es el conjunto de los datos de entrada e Y la salida. El algoritmo aproxima la función f teniendo en cuenta el valor de Y para los valores de X . En el caso de la clasificación, $Y \in K$ donde K es el conjunto de categorías posibles, y en el caso de un problema de regresión, $Y \in \mathbb{R}$.

- **Aprendizaje no supervisado:** consiste en modelos que no conocen las categorías de los datos, es decir, el modelo aprende únicamente en base a los datos de entrada, sin ninguna supervisión. Son modelos basados en la auto-

organización, los algoritmos deducen la estructura de los datos por su cuenta.

Al igual que en el aprendizaje supervisado, existen dos tipos:

- **Clustering**³: consiste en determinar agrupaciones de los datos de entrada. La principal diferencia con una clasificación de los datos, es que el *clustering* tiene en cuenta factores como la distancia (en el espacio) o la similitud entre los valores para agruparlos, pues desconoce las categorías de datos existentes, que son las que intenta encontrar.
- **Asociación**: es un problema que trata de encontrar similitudes o asociaciones entre los valores de los datos. Por ejemplo, en un supermercado, que la gente que compra X también compra Z .
- **Aprendizaje semisupervisado**: son un conjunto de algoritmos que se utilizan cuando el conjunto de datos contiene tanto datos etiquetados como no etiquetados, es decir, datos de los que se sabe su salida y datos de los que no. Este tipo de aprendizaje se da en muchas ocasiones porque la extracción de datos para un problema real no siempre es la ideal, es decir, no siempre se cuenta con los datos etiquetados, o el hecho de etiquetar datos puede no ser factible.
- **Aprendizaje por refuerzo**: aunque es otro tipo más de aprendizaje automático, realmente está fuera del contexto de los otros tipos, pues no se basa en la entrada de datos y la predicción de las salidas. El aprendizaje por refuerzo consiste en un agente que interactúa con un entorno, de forma que recibe refuerzos positivos o negativos de cada interacción y aprende cuál decisión es la correcta.

Si en los inicios de la aplicación del aprendizaje automático al análisis de sentimiento Pang et al. (2002) planteaban la utilización de algoritmos supervisados, en ese mismo año se había realizado el estudio de Turney (2002), que se centraba en la utilización de algoritmos no supervisados. Ambas publicaciones tomaban como datos una serie de reseñas de películas y las clasificaban en positivas o negativas, obteniendo Pang et al. un mejor resultado de 82,7 %, mientras que Turney consiguió un 74 % con su

³agrupamiento

algoritmo no supervisado.

La existencia en muchas ocasiones de datos no etiquetados ha provocado que durante toda la evolución de este campo hayan surgido muchos estudios experimentando con algoritmos no supervisados. Desde el primer enfoque no supervisado de Turney se han realizado muchos trabajos explorando esta vía, siendo uno de los más destacados el de Lin y He (2009), cuya mayor exactitud lograda es de 84.6 %. Esto es bastante positivo, pues a través de un método totalmente no supervisado mejora los resultados de Pang et al. y de Turney.

Sin embargo, partiendo de la base de Pang et al. también se ha realizado mucha investigación sobre modelos supervisados, los cuáles el modelo de Lin no supera. Los propios Pang y Lee, autores de Pang et al. (2002), plantearon un nuevo modelo dos años más tarde en Pang y Lee (2004) con el que obtuvieron una exactitud de 87,2%, mejorando así en un 4,5 % su propio resultado. También el trabajo de Whitelaw, Garg y Argamon (2005) ha tenido importancia, pues consiguió superar todos los anteriormente mencionados con una exactitud de 90,2 % con un modelo muy similar al de Pang y Lee pero que realiza un análisis previo de las actitudes del texto.

**TABLA 2.1. COMPARATIVA DE MODELOS PREVIOS
SEGÚN EL TIPO DE APRENDIZAJE**

Estudio	Tipo de aprendizaje	Mejor resultado
Pang et al. (2002)	Supervisado	82,7 %
Turney (2002)	No supervisado	74 %
Lin y He (2009)	No supervisado	84,6 %
Pang y Lee (2004)	Supervisado	87,2 %
Whitelaw, Garg y Argamon (2005)	Supervisado	90,2 %

El resto de aproximaciones existentes han estado basadas en estas publicaciones mencionadas previamente y son muy similares en cuanto a resultados. En general, observando la literatura de los enfoques supervisados y no supervisados, los primeros

han conseguido siempre mejores resultados. Por este motivo, y teniendo en cuenta que el conjunto de datos de Yelp está etiquetado, es decir, se conoce el sentimiento de cada reseña, y que la dimensión de este trabajo es limitada, se experimentará únicamente con modelos supervisados.

2.2.1. Algoritmos de aprendizaje supervisado

Igual que con los tipos de aprendizaje, dentro del aprendizaje supervisado existen una serie de algoritmos, y en concreto unos que por sus características son más utilizados para clasificación de texto. Cabe diferenciar que dentro de todos los algoritmos, también se diferencian entre los que se usan para problemas de regresión y los que se usan para clasificar.

Los distintos tipos de algoritmos que se utilizan para clasificación supervisada son los siguientes (Thangaraj & Sivakami, 2018):

- Árboles de decisión
- Clasificadores Bayesianos ingenuos
- K vecinos más próximos
- LVQ (Aprendizaje de cuantificación vectorial)
- Máquinas de soporte vectorial (SVM)
- Métodos combinados (*Ensemble*)
- Perceptrón simple
- Redes de neuronas artificiales
- Regresión logística

Cada algoritmo tiene sus propias características y su propio funcionamiento, por lo que la utilización de unos u otros se estudia cada vez que se va a experimentar un modelo, pues dependiendo del tipo de datos de entrada y del tipo de problema unos pueden dar mejor resultado que otros.

Los algoritmos utilizados más frecuentemente son las Máquinas de soporte vectorial (SVM), Regresión logística (en algunos estudios llamada Máxima entropía, pues son modelos equivalentes (Mount, 2012)) y Bayes ingenuo. No obstante, también ha sido habitual la aplicación de Árboles de decisión y métodos combinados (*ensemble*).

Si nos centramos en los modelos explorados en la tabla 2.1, Pang et al. (2002) emplearon Bayes ingenuo, Regresión logística y SVM, obteniendo mejores resultados con este último. De nuevo, Pang y Lee (2004) vuelven a probar con SVM y Bayes ingenuo. Whitelaw et al. (2005), que consiguieron el mejor resultado de los comparados, utilizaron un modelo de SVM.

Parikh y Movassate (2009) realizan un análisis de sentimiento en Twitter, pues es una de las redes sociales más usadas, y llegan a la conclusión de que Bayes ingenuo logra bastante mejores resultados que la Regresión logística. Por otro lado, Saif, He, Fernandez y Alani (2014) utilizaron los mismos algoritmos para clasificar y en su caso la Regresión Logística supera al clasificador de Bayes ingenuo.

En muchas ocasiones se ha visto que las Máquinas de soporte vectorial son los algoritmos que mejores resultados han dado. Kolchyna, Souza, Treleaven y Aste (2015) plantearon una solución utilizando Bayes ingenuo, SVM y Árboles de decisión y concluyeron que el mejor de ellos era SVM.

También a raíz de la idea de que los algoritmos *ensemble* rendían bien en cuanto a clasificación y de que las Redes de neuronas artificiales no habían sido muy exploradas en cuanto a clasificación de texto, Vinodhini y Chandrasekaran (2016) realizaron un estudio comparativo en el cual utilizaban Redes de neuronas, un algoritmo compuesto y SVM, y llegaron a la conclusión de que el que mejor resultados obtiene es el algoritmo compuesto.

Como se ha observado, se utilizan diversos algoritmos de *machine learning* para casos como éste y aunque está claro cuáles son los más aplicados, no hay evidencia sobre cuál da mejores resultados. Esto se debe a que se utilizan distintos conjuntos de datos, se realizan distintos preprocesados sobre los datos o se realizan distintas interpretaciones de los problemas, entre otros factores. Aún así, los distintos estudios coinciden en la representación del texto a través del modelo de *bag-of-words*.

Por tanto, en vista de la discordancia de resultados en la literatura, se plantearán modelos utilizando los distintos algoritmos más empleados y experimentando en busca del que mejores resultados proporcione. En concreto, se estudiarán los clasificadores Bayesianos ingenuos, la Regresión logística, las Máquinas de soporte vectorial y una combinación de Árboles de decisión.

Clasificadores Bayesianos ingenuos

*Naïve Bayes*⁴ es quizá el algoritmo de aprendizaje más simple pero también uno de los más eficientes, por lo que los clasificadores Bayesianos ingenuos son unos de los más empleados en este campo.

Los clasificadores Bayesianos ingenuos son clasificadores estadísticos, pues están fundamentados en el teorema de Bayes, razón por la cuál es necesario comprender este teorema. Raschka (2014) provee un buen estudio sobre este algoritmo de clasificación. El teorema se puede escribir de la siguiente forma:

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)} \quad (2.2)$$

donde

$P(A)$ es la probabilidad a priori de que ocurra A ,

$P(B)$ es la probabilidad de que ocurra B ,

$P(B | A)$ es la probabilidad de que ocurra B si ocurriese A

y $P(A | B)$ es la probabilidad a posteriori.

La propiedad a posteriori en un problema de clasificación se podría interpretar de la siguiente manera si se piensa en A como una categoría de clasificación y en B como el conjunto de los datos de entrada: “¿Cuál es la probabilidad de que un objeto concreto pertenezca a la clase A si se conocen sus valores de entrada B ?”. En realidad, $P(B)$ no es necesaria, pues el conjunto de B son datos dados y $P(B)$ no depende de A , por lo que sería una constante, quedando la fórmula:

⁴Bayes ingenuo

$$P(A | B) = P(B | A) P(A) \quad (2.3)$$

Teniendo en cuenta lo anterior, la probabilidad a posteriori en un problema de clasificación sería $P(A | B_1, \dots, B_m)$ si tenemos en cuenta m de datos de entrada y aplicamos la fórmula a todos ellos. Pero los clasificadores Bayesianos toman el “ingenuo” supuesto de que los valores de entrada son independientes entre ellos dada la clase a la que pertenecen. Bajo este supuesto, la probabilidad es el producto de las probabilidades independientes de cada valor de entrada:

$$P(B | A) = P(B_1, B_2, \dots, B_m | A) = \prod_{n=1}^m (P(B_n | A)) \quad (2.4)$$

La probabilidad independiente $P(B_i | A)$ es la proporción de todos los valores dentro de la categoría A que tienen un valor de B_i , con lo que el teorema se simplifica:

$$P(A | B) = P(A) \prod_{n=1}^m (P(A | B_n)) \quad (2.5)$$

Raschka (2014) no sólo realiza un estudio de los clasificadores Bayesianos, sino también de su aplicación a clasificación de texto.

En el proceso de aprendizaje, el algoritmo únicamente tiene que calcular las probabilidades para cada entrada, por lo que es muy rápido, pero el algoritmo puede flaquear si el supuesto de la independencia de las variables no se da. En el caso de texto, se podría pensar que ciertas palabras que suelen aparecer juntas realmente son dependientes, pero realmente cada palabra es independiente, aunque si un modelo tomase en cuenta palabras que aparecen juntas de forma frecuente obtendría, posiblemente, mejores resultados, lo cuál hay que experimentar.

En general, el algoritmo de Bayes ingenuo obtiene buenos resultados pero es importante realizar una buena selección de las palabras de entrada (aplicación del modelo *bag-of-words*, detallado en la sección 2.2.2). Más allá, existen distintas variantes de clasificadores de Bayes que han sido particularmente efectivas para clasificación de texto. Estas variantes se distinguen en la distribución de la probabilidad y son la

distribución multinomial y la distribución de Bernoulli (Mccallum & Nigam, 2001).

La distribución multinomial funciona muy bien cuando las palabras están representadas como un vector de ocurrencias, aunque también ha dado buenos resultados con vectores de frecuencias. La fórmula es la siguiente, en la cuál $\hat{\theta}_{yi}$ es la probabilidad de una palabra i de aparecer en una entrada de la categoría y :

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \quad (2.6)$$

donde

$N_{yi} = \sum_{x \in T} x_i$ es el número de veces que la palabra i aparece en una entrada de la categoría y en el conjunto de entrenamiento T ,

y $N_y = \sum_{i=1}^n N_{yi}$ es el número total de ocurrencias de las palabras de la categoría y .

La regla según la distribución de Bernoulli sería:

$$P(x_i | y) = P(i | y)x_i + (1 - P(i | y))(1 - x_i) \quad (2.7)$$

Se diferencia de la multinomial en que penaliza la no-ocurrencia de una palabra i que debería pertenecer a una clase y , mientras que la variante multinomial simplemente ignoraría una palabra que no aparece.

En vista de lo explicado, se aplicarán clasificadores Bayesianos ingenuos utilizando la distribución multinomial.

Máquinas de soporte vectorial

Las máquinas de soporte vectorial (SVM) son un conjunto de algoritmos de aprendizaje supervisado basados en *kernels* muy usados en problemas de regresión o clasificación, aunque mayoritariamente en estos últimos. La clave de su funcionamiento está en las funciones *kernel*, que se explicarán posteriormente.

La base teórica de estos algoritmos es algo compleja, por lo que son más complicados de explicar que un algoritmo basado en reglas como es el de *Naïve Bayes*. No obs-

tante, existen muchas publicaciones que explican este algoritmo de forma muy clara, como el artículo de Schölkopf y Smola (2002) o los apuntes de Ng (2017). No se entrará profundamente en la base matemática del algoritmo, pues estas publicaciones ya la explican y no es el objetivo de este apartado.

Estos algoritmos se basan en la representación de los valores de entrada como puntos en un espacio vectorial y la creación de un hiperplano para separar las distintas categorías, como en el siguiente ejemplo:

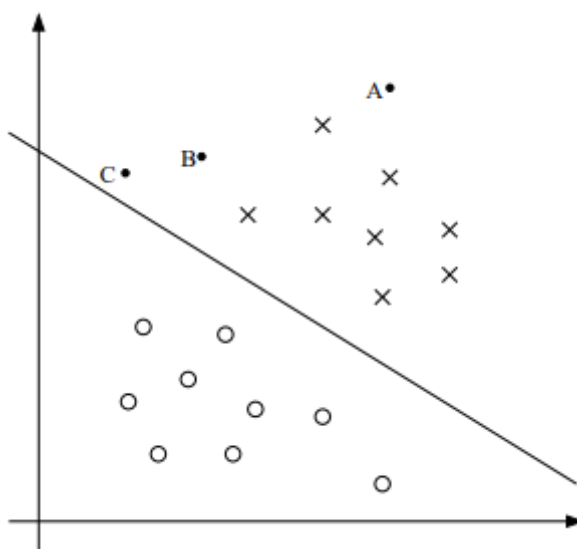


Fig. 2.3. Representación de una clasificación binaria con SVM (Ng, 2017)

En la figura anterior, se tomarían las equis y los círculos como valores de distintas clases, las cuáles el hiperplano de separación diferencia. Sin embargo, si nos fijamos en los puntos A, B y C representados en el espacio, nos encontramos con el primer problema de una SVM. El punto A está claramente en la clase de las equis, pero contrariamente, el punto C está muy cerca del límite, por lo que un ligero cambio en la pendiente podría cambiar la clasificación y meterlo dentro de los círculos. Esto es el problema de optimización de las SVM y consiste en encontrar el hiperplano óptimo que maximice la separación entre las categorías (Ng, 2017).

Teniendo en cuenta los puntos representados, hay infinitos hiperplanos que podrían separarlos, pero al realizar el aprendizaje del modelo no cualquier hiperplano generalizaría bien la diferenciación entre una categoría y otra. Por esto, la mejor solución

posible es el hiperplano que maximice la separación entre categorías:

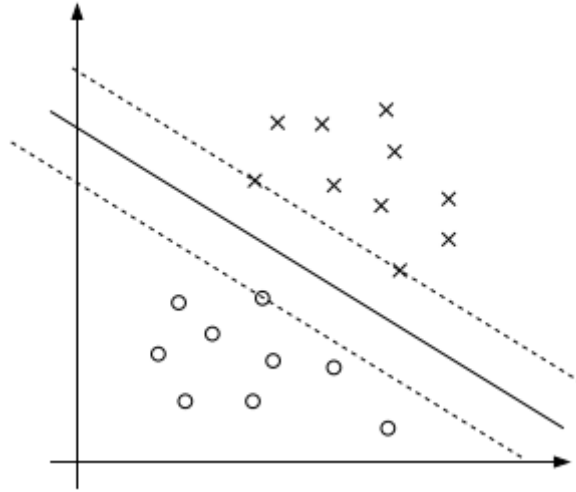


Fig. 2.4. Máximización de márgenes en una clasificación binaria con SVM (Ng, 2017)

La figura 2.4 representa un hiperplano con máxima separación en el cuál los puntos de cada clase con menos distancia al hiperplano están en el límite. Estos puntos que se encuentran a la menor distancia posible del hiperplano son los llamados vectores de soporte y el objetivo es que la distancia entre las líneas paralelas que forman los vectores de soporte sea la máxima posible.

Hasta ahora, la situación que se ha estudiado es una en la cual los valores son linealmente separables, es decir, todos los puntos de las categorías se pueden separar mediante un hiperplano. No obstante, en la práctica no ocurre que los valores sean separables de esta forma, sino que estos valores se encuentren en el espacio de forma que sea necesaria una curva no lineal, se tengan más de dos atributos de entrada o existan más de dos categorías. Es en estos casos donde son necesarias las funciones de *kernel*:

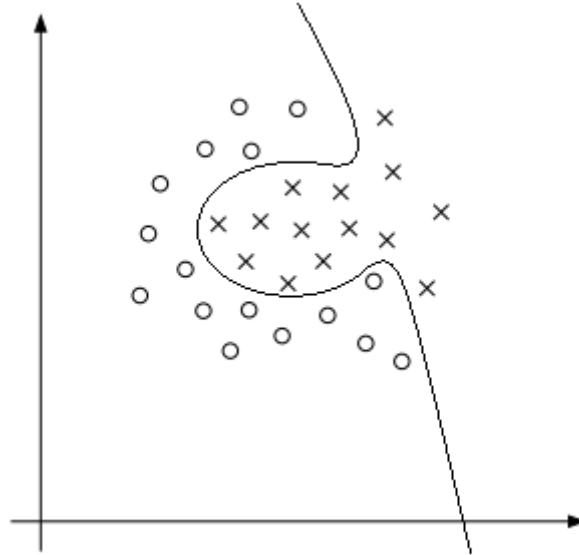


Fig. 2.5. Representación de una clasificación no lineal con SVM

Según se puede leer en Ng (2017), las funciones de *kernel* solucionan el problema anterior transformando el espacio vectorial de entrada en un espacio vectorial de dimensión superior para facilitar la separación lineal de los puntos. Así, partiendo de un conjunto de atributos de entrada x , se trata de encontrar la función $\phi(x)$ que transforme adecuadamente el espacio de entrada:

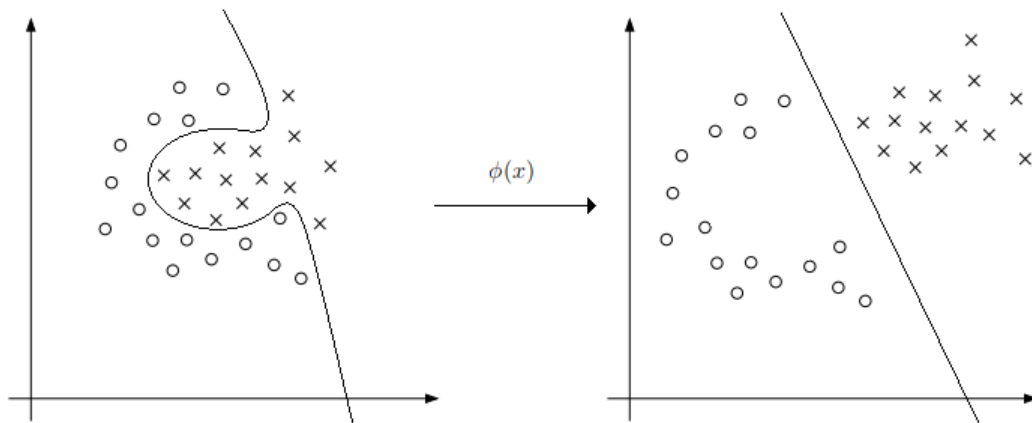


Fig. 2.6. Transformación mediante kernel a un problema no lineal

Existen distintas funciones *kernel*, por lo que la aplicación de unas u otras puede aportar distintos resultados. Entre ellas están la función sigmoide

$$\tanh(\gamma(x, x') + \theta) \quad (2.8)$$

la función polinomial

$$(\gamma(x, x') + \theta)^d \quad (2.9)$$

o la función de base radial Gaussiana

$$\exp(-\gamma \|x - x'\|^2) \quad (2.10)$$

donde

(x, x') es el punto a transformar

y θ y d son parámetros.

Teniendo en cuenta el ajuste del hiperplano, las Máquinas de soporte vectorial generalizan muy bien y clasifican de forma muy eficiente. Además, existe la posibilidad de aplicar una función *kernel* propia, distinta de las habituales, por lo que es un algoritmo bastante versátil.

Una de las características más destacadas es que la capacidad de aprendizaje es independiente de la dimensión del espacio de entrada, lo que significa que permite una buena generalización ante la presencia de muchos atributos de entrada (muchas palabras en el caso de texto). Otra ventaja es que funciona muy bien tanto con vectores de entrada dispersos (donde muchos de los valores son 0) como con vectores densos (todos los valores tienen algún valor), por lo que favorece el problema de clasificación de texto, pues no todas las palabras aparecen en un determinado texto (Joachims, 1998).

Regresión logística

La regresión logística es un análisis de regresión que emplea una función logística para estimar el valor de una variable a partir de una serie de variables independientes.

Un modelo de regresión logística toma el vector de atributos de entrada x de la forma (x_1, x_2, \dots, x_n) y la variable categórica y , junto con un vector de coeficientes β de la forma $\beta = (\beta_1, \dots, \beta_n)$ que modificará el valor del vector de entrada como una combinación lineal $\beta_0 + \beta_1 x_1 + \dots + \beta_n x_n$. Los coeficientes β son parámetros del modelo y β_0 es una constante. Si la regresión toma la función logística, la probabilidad de una entrada de pertenecer a una categoría es (Ifrim, Bakir & Weikum, 2008):

$$p = \frac{e^{\beta^T \cdot x}}{1 + e^{\beta^T \cdot x}} \quad (2.11)$$

El objetivo del modelo es encontrar el vector de parámetros β que maximiza la probabilidad logarítmica de los datos de entrenamiento. La probabilidad logarítmica para este conjunto sería:

$$l(\beta) = \sum_{n=1}^N [y_i \cdot \beta^T \cdot x_i - \log(1 + e^{\beta^T \cdot x})] \quad (2.12)$$

El modelo en su esencia es muy parecido a *Naïve Bayes*, siendo ambos clasificadores lineales basados en la probabilidad que separan las categorías con un hiperplano. Los clasificadores de Bayes funcionan muy bien con conjuntos de datos pequeños mientras que este clasificador lineal es bueno para conjuntos de datos grandes. La regresión logística produce buenos resultados para categorización de texto. En este tipo de problemas, los clasificadores tienen dificultades cuando el número de palabras supera el número de observaciones, pero la regresión logística supera mejor estas dificultades que otros modelos (Thangaraj & Sivakami, 2018).

Random Forest

El aumento de la capacidad de cómputo ha provocado la aparición de algoritmos que realizan una gran cantidad de cálculos, lo que ha resultado en los métodos combinados o de *ensemble*. Estos métodos se basan en realizar el proceso de clasificación simultáneamente con distintos clasificadores (lo cuál implica la mayor necesidad de cómputo) para después combinar los resultados, mejorando la clasificación.

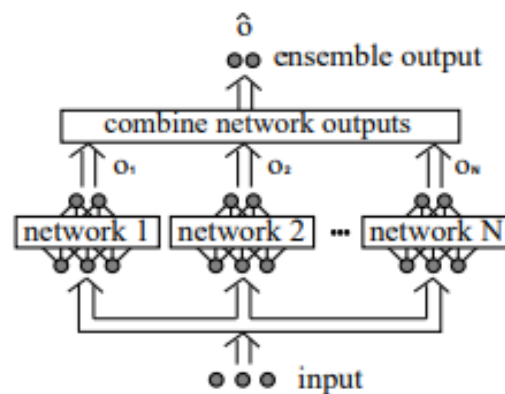


Fig. 2.7. Un clasificador *ensemble* de redes neuronales (Maclin & Opitz, 2011)

De acuerdo a Maclin y Opitz (2011), la combinación de clasificadores solo es útil en las ocasiones en las que los resultados de los clasificadores son distintos entre ellos, pues al buscar la combinación entre todos se puede encontrar una solución mejor. Obviamente, la combinación de clasificadores que han obtenido mismos resultados no es productiva. Este estudio también afirma que la situación ideal es en la que se tienen buenos clasificadores que discrepan lo máximo posible.

La figura 2.7 representa un ejemplo de combinación, en la cual los clasificadores que se combinan son redes neuronales. Estos modelos son muy utilizados en los métodos de *ensemble* porque permiten su alteración con distintas arquitecturas, capas, pesos o parámetros, obteniendo clasificadores completamente distintos. Se distinguen dos tipos de métodos *ensemble*, que tratan de generar discrepancias entre los clasificadores alterando el conjunto de entrenamiento, los métodos de *bagging* y los de *boosting*.

Realmente, sería posible aplicar algoritmos de ambos tipos para este problema de clasificación de texto, pero como se ha mencionado previamente en este documento, la dimensión de un Trabajo Fin de Grado no permite analizar todos, por lo que se revisa la literatura para experimentar con el algoritmo que mejores resultados ha dado.

Entre otros estudios, el propio de Maclin y Opitz (2011), que explica ambos tipos de *ensemble*, también realiza una comparativa entre ellos y llega a la conclusión de que el *bagging* es más apropiado para la mayoría de problemas, pues es más independiente de los datos de entrada, mientras que los métodos de *boosting* tienen más correlación con los datos, por lo que generalmente se ven más perjudicados, aunque en los casos en los que esta dependencia del conjunto de datos es buena, el *boosting* tiene mucho mejor rendimiento.

Bagging es un método de *ensemble* que toma el conjunto de datos inicial y lo separa en m subconjuntos. Estos subconjuntos se forman tomando ejemplos del conjunto de datos de forma aleatoria y con reemplazos, es decir, que se repiten ejemplos en distintos subconjuntos. Partiendo de los m subconjuntos formados se crean m clasificadores que utilizan los diferentes subconjuntos generados. Estos métodos funcionan muy bien con modelos “inestables” en los que cualquier pequeño cambio en el conjunto de datos puede producir grandes cambios en la clasificación, y los más destacados algoritmos “inestables” son los árboles de decisión y las redes neuronales, por lo que el *bagging* ha sido muy aplicado con combinaciones de estos algoritmos. (Maclin & Opitz, 2011).

El algoritmo de *Random forest*, o Bosques aleatorios, es un tipo de algoritmo de *ensemble* que se basa en la composición de árboles de decisión.

Un árbol de decisión es un algoritmo de predicción muy fácil de comprender. De forma simple, se quiere predecir la categoría Y en base a una serie de entradas X_1, X_2, \dots, X_n , para lo cuál se construye la estructura jerárquica de un árbol. En este árbol, cada nodo interno, partiendo de un nodo raíz, representa una toma decisión sobre un atributo de entrada X_i . De un nodo interno salen ramas hacia otros nodos internos, que representan la decisión tomada sobre ese nodo interno. Los nodos que

no se dividen más son hojas y representan una categoría de Y a la cuál pertenecería un valor de entrada que ha recorrido esa rama terminando en la hoja (Loh, 2011). En la siguiente figura se puede apreciar un árbol que representa un problema con 3 categorías y 2 variables X :

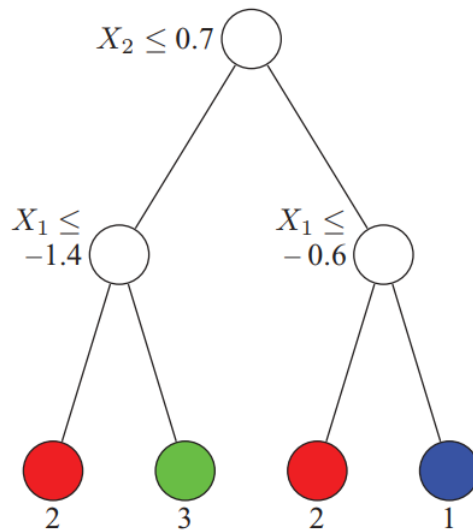


Fig. 2.8. Un árbol de decisión para clasificación (Loh, 2011)

En cada nodo interno, la decisión se va por la rama izquierda sólo si se da la condición del nodo, y en caso contrario toma la rama de la derecha.

La construcción de este árbol se puede hacer de distintas maneras, por lo que existen distintos algoritmos para ello. Esta construcción debe decidir las variables de entrada y las condiciones que se usan para realizar las divisiones, así como en qué momento un nodo no va a dividirse (va a ser una hoja). Los principales algoritmos para la construcción de árboles son ID3, C4.5 y CART (Loh, 2011).

Teniendo en mente de forma general el funcionamiento de los árboles de decisión, se pueden explicar los algoritmos basados en la composición de los mismos, es decir, *Random forest*.

Random forest es un algoritmo de *bagging*, y como se explicó, estos métodos toman subconjuntos de datos de forma aleatoria y con reemplazos. El método consiste en construir un modelo de árbol para cada uno de estos subconjuntos y entrenarlo.

Una vez han sido entrenados, se calcula la moda (la media en el caso de árboles de regresión) para clasificar los datos a predecir. La composición de distintos modelos permite al algoritmo obtener mejores resultados, reduciendo la varianza del modelo sin aumentar el sesgo de las predicciones. Esto ocurre porque el entrenamiento de un solo árbol es sensible a la existencia de ruido en el conjunto de entrenamiento, y lo mismo ocurre si distintos árboles emplean el mismo conjunto, pero al utilizarse subconjuntos, los árboles no están correlacionados, lo que da lugar a buenos resultados (Breiman, 2001).

Sin embargo, partiendo de la idea de *bagging*, la idea de la composición de árboles ha ido más allá. Ho (2002) distinguió al algoritmo de *Random forest* de la idea general de *bagging* utilizando un subconjunto aleatorio pero de las variables de entrada. El motivo es que aunque la correlación de los árboles se ve reducida si se utilizan distintos subconjuntos, puede ocurrir que algunas de estas variables tengan mucho peso en la predicción de las categorías, lo cual provoca que muchos de los árboles construidos utilicen estas variables, creando una nueva correlación. Normalmente, si se cuenta con k variables de entrada, cada subconjunto separado cuenta con \sqrt{k} (redondeado hacia abajo) variables de entrada.

2.2.2. El modelo bag-of-words

Previamente se ha explicado el problema de la clasificación de sentimiento y los distintos algoritmos aplicables desde el aprendizaje automático, pero existe otra variable dentro de este problema que tiene más peso y es la representación del texto.

En el comienzo de la historia de la computación existió un problema: los ordenadores solo entendían números (más concretamente ceros y unos). Este problema se resolvió, pero se puede trasladar al campo del aprendizaje automático: los algoritmos de aprendizaje automático toman vectores de números como entrada y su salida son también números. En el contexto de un problema de clasificación de texto, la entrada al algoritmo sería el texto a clasificar y su salida sería la categoría del texto, con lo que se debe encontrar una forma de representar el texto como una serie de números.

La representación del texto como palabras es, al final, una tarea de ingeniería de características o *feature engineering*. De acuerdo a Domingos (2012) y a la literatura general, en este área, se llama característica o *feature* a cada propiedad individual de un fenómeno observado, es decir, a cada variable de entrada en un algoritmo de aprendizaje automático. Muchos modelos de aprendizaje obtienen buenos resultados y otros, al contrario, no, y la diferencia entre ellos es el factor que más importancia tiene: la extracción de estas características.

La extracción de características es un proceso esencial en un modelo de aprendizaje automático que consiste en crear las características de entrada para que un algoritmo funcione. En la clasificación de texto, el principal modelo de ingeniería de características es el modelo de *bag-of-words*.

Aunque este modelo fue planteado por primera vez en Harris (1954), no fue hasta la publicación de Pang et al. (2002) cuando se comenzó a emplear en *machine learning* para clasificar texto. Desde entonces, ha sido el modelo utilizado generalmente en la literatura para clasificación de sentimiento, como por ejemplo en publicaciones como Ifrim et al. (2008), Kolchyna et al. (2015) o van Atteveldt, Kleinnijenhuis, Ruigrok y Schlobach (2008).

La principal idea de este modelo se obtiene de su propio nombre, “bolsa de palabras”, y es transformar cada palabra en un número de forma que la entrada del algoritmo de clasificación es un vector de números, una bolsa de palabras, en el que cada posición es una palabra del texto a clasificar. Para realizar esta transformación existen dos posibles formas: transformar en un vector de de ocurrencias o transformar en un vector de frecuencias.

La razón de que se llame bolsa es que no tiene en cuenta el significado de cada palabra ni su contexto, es decir, toma cada palabra como un número independiente y solo tiene en cuenta si la palabra aparece o no aparece en el texto. Esto es una de las principales desventajas del modelo, pues el considerar la semántica de las palabras puede beneficiar al resultado.

A priori el modelo es simple, pero su capacidad va más allá. En lugar de tomar cada palabra para construir los vectores, se podría tomar cada carácter o cada cierto

número de caracteres. Otra posibilidad, muy usada y quizá la más potente, es el uso de n-gramas, siendo un n-grama una secuencia de elementos. Veamos este modelo con la siguiente frase de ejemplo:

Me ha gustado mucho este restaurante.

Partiendo de esta frase, utilizando el modelo de *bag-of-words*, que en realidad serían unigramas, se obtendría la siguiente lista:

["Me", "ha", "gustado", "mucho", "este", "restaurante"]

Pero si se utilizase un modelo de n-gramas, por ejemplo, bigramas, quedaría la siguiente lista:

["Me ha", "ha gustado", "gustado mucho", "mucho este", "este restaurante"]

La potencia de este modelo reside en que en vez de construir el vocabulario en base a palabras, se forma en base a n-gramas, lo que permite emplear el modelo de *bag-of-words* pero dándole a las palabras cierto contexto, lo cual mejora los resultados en clasificación de texto a costa de tiempo de cómputo.

Bag-of-words tiene algunas limitaciones más. La primera de ellas es que hay que realizar un procesamiento del texto para reducir el tamaño del vocabulario, de modo que tenga en cuenta únicamente las palabras realmente útiles y sea manejable. Y otra es el uso de vectores dispersos. Un vector es disperso cuando la mayoría de sus elementos es cero. Esto ocurre en este modelo porque al tener muchos textos en cuenta el vocabulario que se construye cuenta con todas las palabras, mientras que en cada texto individual aparecen solo una pequeña cantidad de ellas, lo que da lugar a que cada texto individual es representado como un vector con una gran cantidad de ceros.

Antes de convertir el texto en vectores, se debe procesar el mismo para construir un buen vocabulario, pero este tema se tratará en el siguiente capítulo. A continuación, se describen las dos formas de transformar el texto.

Transformación en vector de ocurrencias

La manera más simple de transformar el texto a un vector es a través del número de ocurrencias de cada palabra. Esto se ve mejor con dos frases de ejemplo:

Me ha gustado mucho este restaurante.

Este hotel está bien, me ha gustado, y su situación está bien también.

En base a las frases de ejemplo, se construye el siguiente vocabulario:

{“Me”, “ha”, “gustado”, “mucho”, “este”, “restaurante”, “hotel”, “está”, “bien”, “y”, “su”, “situación”, “también”}

Y teniendo en cuenta el vocabulario formado, se construye el vector de entrada para cada frase según las ocurrencias de cada palabra:

[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

[1, 1, 1, 0, 1, 0, 1, 2, 2, 1, 1, 1, 1]

Siendo el resultado el vector de entrada al algoritmo, en el cuál se puede ver representado el principal problema de este modelo, la dispersión. Si se tuviera un vocabulario más amplio, lo cuál ocurre en la realidad, la mayoría de las posiciones del vector serían ceros.

Esta transformación tiene un problema: solo le da importancia a la aparición de una palabra en el texto, lo que provoca que se le de más peso a los textos largos que a los textos con menos palabras. Dependiendo de las características del conjunto de datos y del modelo, esta representación puede funcionar, pero existe otra forma de realizar esta transformación: la frecuencia inversa.

Transformación en vector de frecuencias inversas

El problema de tener en cuenta sólo las ocurrencias es que palabras que aparecen con mucha frecuencia llegan a tener demasiada importancia, cuando realmente es posible que no contengan información determinante para la clasificación del texto.

Para evitar esto, se debe modificar el valor que se le da a cada palabra, reduciendo la importancia de palabras que aparezcan en muchos textos.

Esta solución se realiza a través del TF-IDF, del inglés *Term frequency - Inverse document frequency*, que en español sería frecuencia de término - frecuencia inversa de documento, que es una medida muy utilizada en recuperación de información. Leskovec, Rajaraman y Ullman (2014) realizan una buena descripción del funcionamiento de esta medida.

Si tenemos términos (palabras) t y documentos (textos) d , la frecuencia de un término t en un documento d , $tf(t, d)$ no es más que el número de ocurrencias del término t en el documento d .

La frecuencia inversa de un término t es la frecuencia de ese término en el resto de textos y se calcula de la siguiente manera:

$$idf_t = \log\left(\frac{N}{n_t}\right) \quad (2.13)$$

donde

N es el número total de textos con los que se cuenta

y n_t es el número de textos en los que aparece el término t .

Al realizar la transformación de un texto a un vector utilizando esta medida, el valor que se le da a cada palabra en el texto es el producto de la frecuencia y la frecuencia inversa, $tf(t, d) \cdot idf_t$.

De esta forma, si se toman como ejemplos las frases descritas en el método anterior, el resultado del vector sería el siguiente:

[0, 0, 0, 0,3, 0, 0,3, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0,3, 0,6, 0,6, 0,3, 0,3, 0,3, 0,3]

Se puede apreciar cómo el valor de las palabras que aparecen en ambas frases se ha visto reducido.

En la realidad, ambos métodos funcionan bien, pues todo depende del conjunto de

datos y el algoritmo que se utilice. Si se experimenta con ambos métodos y con las posibilidades de extensión del modelo (n-gramas o vectorización de caracteres), *bag-of-words* puede dar resultados muy buenos y a su vez muy variados.

2.3. Word embeddings

Como se ha visto, los algoritmos de aprendizaje automático son incapaces de procesar texto plano, por lo que surgió el modelo de representación de texto que se ha visto en la sección anterior. Aunque este modelo ha permitido alcanzar muy buenos resultados, el hecho de tener en cuenta las palabras de forma independiente y sin tener en cuenta su contexto o su significado ha provocado que la investigación en este área haya continuado en búsqueda de un modelo que incorpore estas características.

A pesar de que esta búsqueda ha estado presente desde los comienzos de este área, el aumento de la capacidad de cómputo ligado al desarrollo de distintas arquitecturas de redes neuronales y del aprendizaje profundo ha permitido la aparición de nuevos modelos que tengan en cuenta la semántica y el contexto de las palabras, mejorando los resultados al considerar las relaciones que existen entre las palabras. El propio estudio de la semántica distribucional de Harris (1954), que por primera vez mencionó el modelo *bag-of-words*, estipulaba también que las palabras que suelen aparecer juntas en el mismo contexto expresan significados similares.

Se han logrado distintos modelos que interpretasen la idea de tener en cuenta el contexto de las palabras, como la utilización de una matriz de co-ocurrencia, en la cuál se representan todas las palabras del vocabulario y se cuentan las ocurrencias de cada par de palabras que aparecen juntas. No obstante, la verdadera explosión de esta forma de representación ha venido de la mano de la exploración de distintas arquitecturas de redes neuronales artificiales, por lo que es necesario tener una idea general sobre su funcionamiento.

Las redes de neuronas artificiales son modelos computacionales (del campo del *machine learning*) basados en el funcionamiento de las neuronas biológicas, pues nacen del objetivo de encontrar un algoritmo que procese la información de forma similar al cerebro humano: las dendritas reciben información mediante estímulos, la procesan

y la propagan a otras neuronas a través del axón. Zurada (1992) provee una buena introducción a este tipo de modelos.

Las neuronas artificiales imitan este funcionamiento: obtienen una entrada, modifican su estado de acuerdo a esa entrada y producen una salida. La red de neuronas se forma a través de la unión de varias neuronas, conectando las salidas de unas a las entradas de otras y formando un grafo ponderado y dirigido. En la figura 2.9 se puede apreciar un ejemplo de estructura de una red neuronal. Esencialmente, se cuenta con una capa de neuronas de entrada, que representan cada una de las variables de entrada, y una capa de salida en las que se encuentran las neuronas con el resultado del modelo. Entre estas capas se pueden encontrar capas ocultas, que son las que contienen las neuronas que realizan el procesamiento de los datos de entrada (Zurada, 1992).

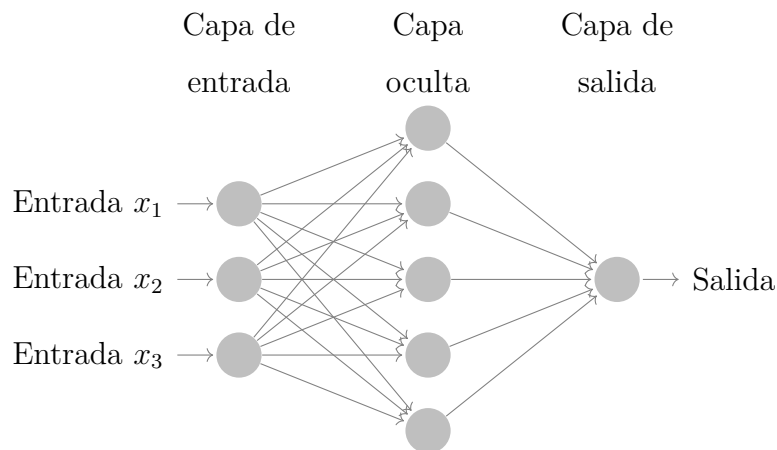


Fig. 2.9. Estructura de una red neuronal artificial

Vista la arquitectura general de una red neuronal, es necesario comprender el funcionamiento del modelo. Cada neurona i tiene un estado, un valor, llamado activación α_i y la unión entre una neurona i y otra neurona j tiene un peso w_{ij} . La activación de las neuronas de entrada corresponde a los valores de los datos de entrada y estos valores se propagan al resto de capas de la red a través de una función de propagación:

$$\alpha_j = \sum_i \alpha_i w_{ij} \quad (2.14)$$

donde

α_j es la activación de la neurona j ,

α_i es la activación de la neurona i ,

y w_{ij} es el peso de la unión entre las neuronas i y j .

El valor de la salida de la red neuronal se calcularía de la misma forma, pues sería el cálculo de la activación de las neuronas de salida. En ocasiones, se calcula la salida de la red aplicando una función f de activación distinta de la función identidad ($f(x) = x$), como la función sigmoideal, la arcotangente o la ReLU. Además, sobre este cálculo se añade un valor de umbral θ , que funciona como un peso constante. Este funcionamiento se puede ver representado en la siguiente figura:

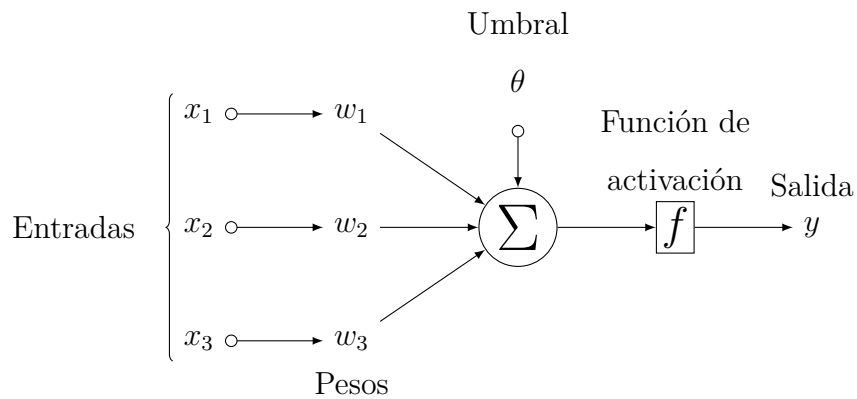


Fig. 2.10. Esquema del modelo de una red neuronal artificial

El aprendizaje de una red de neuronas se basa en determinar los pesos apropiados a partir de un conjunto de ejemplos. Para ello, cada modelo de red neuronal cuenta con una ley de aprendizaje. En el proceso de aprendizaje se recibe un conjunto de ejemplos y se itera sobre cada uno de ellos. Sobre cada ejemplo, se realiza la propagación de los valores para calcular la salida de la red y con esta salida se calcula la diferencia con la salida esperada del ejemplo, obteniendo el error de la red. Partiendo de este error y en base a la ley de aprendizaje del modelo concreto, se realiza la *propagación hacia atrás*, que consiste en modificar todos los pesos de la red para que el resultado se acerque más al esperado. En este proceso también se tiene en cuenta un valor llamado *tasa de aprendizaje* que expresa cuánto se modifican los pesos de la red (Zurada, 1992).

Los distintos modelos existentes varían en la utilización de distintos algoritmos de aprendizaje, distintas funciones de activación, distintas modificaciones de la tasa de aprendizaje o distintas arquitecturas (número de capas y número de neuronas por capa).

El desarrollo de nuevas arquitecturas de redes neuronales ha tenido impacto también en el área de la clasificación de sentimiento pero ha sido a partir de la exploración del uso de *word embeddings* (en español, “incrustaciones de palabras”) cuando se han convertido en modelos clave, debido a que se ha visto que funcionan realmente bien para esta tarea. Los *word embeddings* son una serie de enfoques que buscan representar texto teniendo en cuenta el contexto de las palabras a través de vectores “densos”, es decir, solucionan el problema del modelo *bag-of-words* que utilizaba vectores dispersos, pues cada vector representaba todas las palabras del vocabulario y la mayoría de sus elementos son ceros, ya que los textos solo incluyen algunas palabras. La idea de estos modelos es representar cada palabra como un vector en un espacio vectorial: a partir de los ejemplos de texto, el modelo aprende la posición que debe ocupar en el espacio fijándose en las palabras que aparecen alrededor. El *embedding* (incrustación) es la representación de una palabra en el espacio vectorial. Así, un texto se transforma en un vector denso porque los únicos elementos que contiene son las palabras del mismo y el contexto se tiene en cuenta porque las palabras que más cercanas se encuentran en el espacio vectorial son las que tienen significados similares.

La explosión de este tipo de modelos se produce con la aparición del modelo *Word2Vec* desarrollado por Mikolov, Sutskever, Chen, Corrado y Dean (2013b), un grupo de ingenieros de Google, lo que provocó un cambio drástico respecto a los modelos previos gracias al aumento de la eficiencia del entrenamiento con redes neuronales. Tal ha sido su importancia que es el modelo de referencia.

No obstante, a partir de la aparición de *Word2Vec*, aunque antes también existían, han surgido nuevos modelos con el mismo objetivo de representar las palabras según su semántica y entre ellos hay alguno que ha tenido mucha importancia. Los investigadores de la Universidad de Stanford Pennington, Socher y Manning (2014) plantearon, tomando como ejemplo al grupo liderado por Mikolov, el modelo *GloVe*

(Global Vectors), un algoritmo de aprendizaje no supervisado que obtiene representaciones de palabras en vectores a través de estadísticas de co-ocurrencia. Asimismo, más tarde apareció el modelo *fastText* de la mano de un grupo de investigadores de Facebook al cuál también pertenece el propio Tomas Mikolov. Este modelo (Bojanowski, Grave, Joulin & Mikolov, 2016) es una extensión de *Word2Vec* que toma en cuenta la morfología de las palabras, representando las palabras como una suma de sus componentes para que el algoritmo considere los efectos de sufijos, prefijos y demás componentes morfológicos.

Estos modelos son los que más impacto han tenido en el área y al ser una evolución del *bag-of-words*, se quiere experimentar con los *word embeddings* para comparar resultados. No obstante, por las limitaciones previamente mencionadas de un Trabajo Fin de Grado, no se pueden probar todos los modelos posibles de este tipo, motivo por el cuál se ha decidido emplear el modelo *Word2Vec*, dado que es el modelo principal y si se observan los resultados en sus publicaciones, los tres obtienen precisiones similares.

El punto común que tienen estos modelos es que, apreciando los resultados de Mikolov et al. (2013b), Pennington et al. (2014) y Bojanowski et al. (2016), los *word embeddings* obtienen mejores resultados que el modelo *bag-of-words*. A pesar de ello, se seguirá experimentando con este último, pues sirve para utilizarlo como modelo base y comparar resultados.

2.3.1. Word2Vec

Word2Vec es, como se ha apuntado previamente, un modelo de *word embeddings* que emplea redes neuronales para representar palabras de un texto en un espacio vectorial. El modelo fue desarrollado por Mikolov et al. (2013b) y está compuesto por dos métodos distintos para la formación de los vectores: *Continuous Bag of Words* y *Skip-gram*.

El modelo de *Continuous Bag of Words*, descrito por Mikolov, Chen, Corrado y Dean (2013a), consiste en tomar el contexto de cada palabra como una entrada y que la red trate de predecir la palabra que corresponde a ese contexto, para lo cuál

se emplea una arquitectura con una única capa oculta. En esta arquitectura, la capa de entrada tiene un tamaño V , que es el tamaño del contexto de la palabra, es decir, del vector del texto en el que aparece la palabra. Este texto se vectoriza de la misma forma que en el modelo *bag-of-words*, con unos y ceros. Por ejemplo, si se cuenta con la frase “Este hotel me gusta” y se quiere representar la palabra hotel, el vector de entrada sería $[0, 1, 0, 0]$. La capa de entrada está conectada a una capa oculta de tamaño N , donde N son las dimensiones en las que se quiere representar la palabra y es un parámetro a ajustar en el modelo, formándose una matriz de pesos W de tamaño $V \times N$. La capa oculta únicamente procesa los valores de las entradas hacia la capa de salida, no hay ninguna función de activación particular, solo una activación lineal:

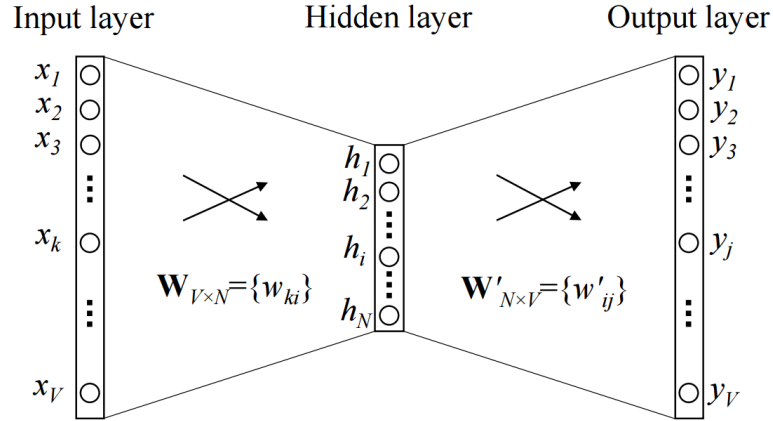


Fig. 2.11. Continuous Bag of Words simple (Usuario *Moucrowap* en Wikimedia)

Según Mikolov et al. (2013a), el modelo toma este nombre debido a que el modelo es una bolsa de palabras en el sentido de que el orden de las palabras no tiene importancia y se diferencia del modelo *bag-of-words* en que usa una representación distribuida de forma continua del contexto.

En la figura 2.11 se ve la arquitectura explicada, pero ésta es la representación de una palabra con un solo contexto. Para lograr una buena representación de las palabras el modelo se entrena empleando una gran cantidad de contextos en los que aparezca la palabra que se quiere representar. Esto se hace utilizando como entradas todos los C contextos de la palabra y calculando en la capa oculta una media de los valores

de entrada:

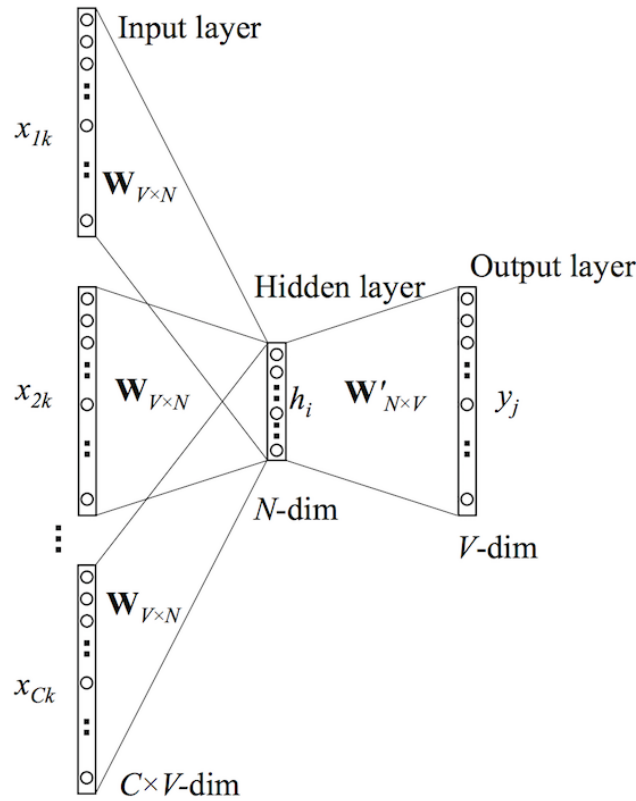


Fig. 2.12. Continuous Bag of Words con múltiples contextos (Usuario *Moucrowap* en Wikimedia)

De esta manera, se producen representaciones de palabras a partir de su contexto, pero Mikolov introdujo otra variante, *Skip-gram*, que emplea la misma arquitectura pero de forma contraria, trata de predecir el contexto partiendo de la palabra que se quiere representar produciendo esta representación en el proceso:

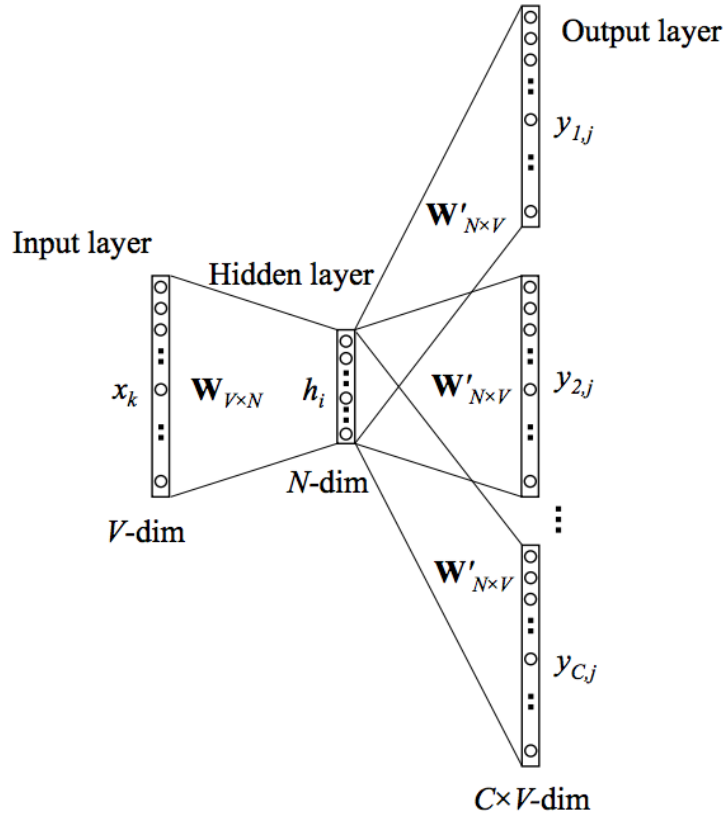


Fig. 2.13. Funcionamiento de Skip-gram (Usuario *Moucrowap* en Wikimedia)

A priori puede parecer que se vuelven a tener vectores dispersos, pero esto es únicamente en la entrada. La fortaleza de este modelo está en la matriz de pesos. Al multiplicar el vector de entrada por la matriz de pesos, el resultado es el vector del elemento que tiene un uno, pues el resto son ceros. Así, se tienen vectores completamente densos.

De acuerdo a Mikolov, *Continuous Bag of Words* es más rápido pero *Skip-gram* produce mejores resultados para palabras poco frecuentes y en casos con poca cantidad de datos.

Para visualizar la potencia de este modelo, Mikolov et al. (2013a) provee un buen ejemplo sobre la capacidad del modelo de comprender la similitud. Sabemos que *grande* es similar a *grandísimo* de la misma forma que *pequeño* es similar a *pequeñísimo*, es decir, no son similares en significado, sino en la relación entre las palabras. Partiendo de esta idea, se le puede preguntar al modelo qué palabra

es similar a *pequeño* de la forma que *grande* es similar a *grandísimo*. Mediante una operación algebraica sobre los vectores de cada palabra se podría computar $X = \textit{grandísimo} - \textit{grande} + \textit{pequeño}$ y un modelo con vectores bien entrenados resultaría que $X = \textit{pequeñísimo}$. Vectores bien entrenados pueden averiguar relaciones entre palabras como por ejemplo un país y su capital: Francia es a París lo que Alemania es a Berlín.

Este modelo obtiene muy buenos resultados para clasificación de sentimiento, pero también se ha aplicado a otros campos en los que ha funcionado muy bien. El mismo Mikolov asegura que “*word2Vec* funciona muy bien cuando se aplica a traducción automática, obteniendo una exactitud superior al 90 % traduciendo de Inglés a Español”. Respecto al campo que nos interesa, Mikolov no plantea la evaluación de los resultados de este modelo a través de su aplicación a la clasificación de sentimiento, sino construyendo un conjunto de relaciones como las del párrafo anterior (capital-ciudad, por ejemplo) y midiendo la exactitud con esas relaciones. En Altszyler, Sigman y Slezak (2017) se realiza una comparación de este modelo con un modelo propio y se llega a la conclusión de que *word2Vec* obtiene mejores resultados cuando se puede entrenar con un gran conjunto de datos de millones de palabras, pero cuando se cuenta con un conjunto más pequeño su precisión disminuye. X. Zhang, Zhao y LeCun (2015) estudian el modelo aplicándolo a clasificación de texto y su mejor resultado alcanza una exactitud de 95,6 %, lo que supone una muy buena exactitud, mejor que la que se obtenía con el modelo *bag-of-words*, cuyos modelos base se comparan en la Tabla 2.1.

2.3.2. Transferencia de aprendizaje y aprendizaje profundo

A la hora de aplicar el modelo visto en un problema real existe una complicación: se quieren representar las palabras en forma de vectores en un espacio vectorial pero, según Mikolov, para obtener buenas representaciones es necesario entrenar los modelos con conjuntos de datos de miles de millones de palabras y formando vectores de grandes dimensiones (incluso hasta 300 dimensiones). Cuando se quiere emplear este modelo para un problema de clasificación de sentimiento como el de este trabajo, resulta prohibitivo entrenar un modelo capaz de obtener buenos resultados,

tanto porque el conjunto de datos con el que se cuenta no tiene el tamaño necesario como porque el tiempo de cómputo y de preparación del conjunto necesario para ello es demasiado.

Es en este contexto en el que entran en juego la transferencia de aprendizaje (*transfer learning* en la literatura anglosajona) y el aprendizaje profundo.

Cuando se trata de obtener un buen modelo de aprendizaje automático, uno de los problemas es la generalización, esto es, la capacidad de ser aplicable a otros datos diferentes de los que se han utilizado para realizar el entrenamiento. Si se obtiene un modelo bien entrenado y con buena capacidad de generalización, este modelo podría ser utilizado en otro problema que utilice el mismo dominio en el cuál los datos no sean fáciles de obtener. En esto consiste la transferencia de aprendizaje. La posible desventaja de este método es la falta de similitud entre los datos del modelo ya entrenado y los del modelo que se quiere entrenar. Aunque ambos dominios sean el mismo, es posible que los datos difieran demasiado y el modelo no obtenga buenos resultados.

En el caso de los *word embeddings*, al no haber posibilidad de entrenar un modelo de representación de palabras en vectores, se emplean vectores entrenados previamente. Los creadores de *word2Vec* y de los demás modelos de *word embedding*, proveen vectores ya entrenados a través de sus propios modelos, los cuáles se pueden utilizar en un problema nuevo (el nuestro).

Estos *word embeddings* son entrenados en estos modelos mediante redes neuronales simples, pero para su aplicación se emplean redes neuronales profundas, es decir, aprendizaje profundo. Éste consiste, definido de forma muy básica, en el uso de redes de neuronas artificiales cuyas arquitecturas cuentan con múltiples capas ocultas que realizan numerosos procesamiento sobre los datos de entrada. La transferencia de aprendizaje encajaría en una arquitectura de aprendizaje profundo mediante una capa oculta de procesamiento en la cuál se encuentran los vectores que representan las palabras del modelo ya pre-entrenado. Al introducirse el vector de un texto a través de la capa de entrada, éste se multiplicaría por los pesos de los vectores ya entrenados y que están en la capa oculta.

Conviene conocer los conceptos generales del aprendizaje profundo para comprender las diferentes arquitecturas en las que se incluye una capa oculta de incrustación de los *word embeddings*.

De las múltiples capas de procesamiento con las que cuenta una arquitectura de aprendizaje profundo, las primeras capas se encargan de procesar los datos de entrada y aprender características simples, mientras que las capas más avanzadas tratan de extraer características más complejas. Por esto, los algoritmos de aprendizaje profundo se emplean en problemas donde la clave está en aprender representaciones de datos. Así, ha sido un conjunto de modelos que han sido muy aplicados en problemas como reconocimiento de imágenes y en los últimos años, gracias a la aparición de los *word embeddings* han tomado mucha importancia en el campo del procesamiento del lenguaje (L. Zhang, Wang & Liu, 2018).

Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) son redes inicialmente empleadas en visión por computador, por lo que están basadas en el córtex visual del cerebro. Esta parte del cerebro contiene células llamadas campos receptivos que se encargan de detectar luz en sub-regiones de los campos visuales. Estas células hacen de “filtro” sobre el espacio de entrada. Este tipo de redes consisten en múltiples capas convolucionales, cada una de las cuáles realiza la función de procesamiento de las células del córtex visual: toman sub-regiones del espacio de entrada y las procesan (convolución). En un problema, se realiza la convolución mediante la aplicación de múltiples filtros (múltiples capas convolucionales) que extraen las características del espacio de entrada, obteniendo lo que se llama *mapa de características o mapa de activación* (L. Zhang et al., 2018).

Por otro lado, estas arquitecturas cuentan con otro tipo de capas: capas de reducción de muestreo o *pooling*. Estas capas se introducen entre capas convolucionales y tratan de reducir el tamaño espacial de las representaciones para reducir la complejidad del entrenamiento.

Al final de una CNN, tras las capas convolucionales y de reducción, se encuentran

neuronas tradicionales, completamente conectadas, que realizan la tarea de clasificación. En la figura 2.14 se puede observar un ejemplo de red convolucional con diversas capas de convolución y de reducción.

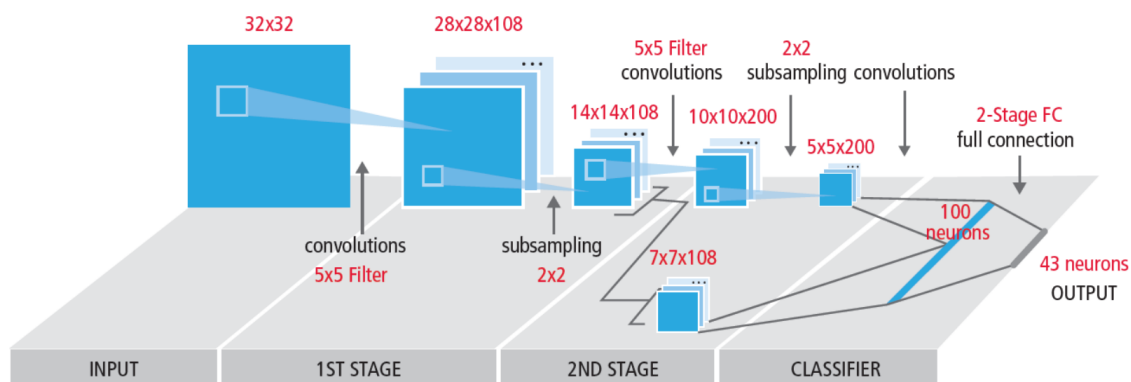


Fig. 2.14. Arquitectura de una red neuronal convolucional (L. Zhang et al., 2018)

Como las capas convolucionales extraen las características más relevantes localmente, este tipo de redes tienen una correlación espacial y local al imponer un patrón de conectividad local entre las neuronas de capas adyacentes. Esto es útil para clasificación de texto, pues en este problema se trata de encontrar “pistas” fuertes que indiquen la pertenencia a una categoría u otra. El objetivo es, a través de este tipo de redes, aprender cuáles palabras son buenos indicadores de la categoría, lo cuál se logra a través de estas capas de convolución y de reducción.

Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN) son un tipo de redes neuronales cuyas conexiones forman ciclos. A diferencia del resto de redes de neuronas, en las cuáles las activaciones se propagan desde las entradas hacia las salidas, en este tipo de redes los ciclos propagan también las activaciones desde la salida hacia atrás, permitiendo a las neuronas conocer lo que ya ha sido aprendido. Se podría imaginar que una RNN es una serie de copias de una misma red dónde cada una de estas copias pasa un mensaje a la siguiente. Por ejemplo, si se trata de procesar un texto de cuatro palabras, una red neuronal recurrente se separaría en una red con cuatro estados,

uno para cada palabra.

L. Zhang et al. (2018) explica su funcionamiento y es relativamente simple. En cada iteración t del ciclo, la red se encuentra en un estado h_t . Si se tiene un vector de entrada x_t y una matriz de pesos W^{hx} , el estado de la neurona sería el producto de ambos, pero como la idea de este tipo de arquitecturas es el uso de lo que la red ya ha aprendido, es decir, que tengan memoria, el estado de la neurona debe tener en cuenta los pesos del estado anterior:

$$h_t = f(W^{hh} \cdot h_{t-1} + W^{hx} \cdot x_t) \quad (2.15)$$

donde la función f es alguna de las funciones de activación ya mencionadas previamente, aunque en este tipo de redes se suelen emplear la función arcotangente o la función ReLu.

Se han desarrollado numerosos tipos de modelos más sofisticados a partir de este modelo estándar: las redes neuronales recurrentes bidireccionales y las redes de gran memoria a corto plazo.

Las RNN se basan en la idea de que la salida en cada estado no solo depende de la salida del estado anterior, sino también de los siguientes, por lo que tiene en cuenta ambas direcciones. Por ejemplo, en el caso de una palabra, el contexto de la misma no viene dado únicamente por su palabra predecesora (lo que sería el estado anterior de la red), sino también de la siguiente palabra en el texto. Una red bidireccional está compuesta por dos redes recurrentes: una procesa la entrada en orden y la otra procesa la entrada revertida, para después calcular la salida en base al estado de ambas redes (L. Zhang et al., 2018).

Redes de gran memoria a corto plazo

Las redes neuronales recurrentes tienen un principal problema (que también puede existir en las demás arquitecturas): el algoritmo de descenso por gradiente, que en redes neuronales se emplea para encontrar el mínimo de la función del error de una red, no funciona bien cuando la red tiene muchas capas porque al hacer la

propagación hacia atrás el valor del gradiente se disminuye cada vez más y al llegar a las primeras capas éste se ha reducido demasiado, de forma que estas capas no aprenden rápido. En el caso de las RNN, cada estado de la red funciona como una capa oculta, lo que da lugar a la situación descrita: demasiadas capas ocultas. Si la idea de las RNN era que tuvieran “memoria” de lo ya aprendido, este problema se traduce en que solo tienen memoria de la información más reciente.

Las redes de gran memoria a corto plazo (LSTM, Long short-term memory) son un tipo de redes neuronales recurrentes capaces de solucionar este problema manteniendo memoria largo plazo de lo que la red considera interesante.

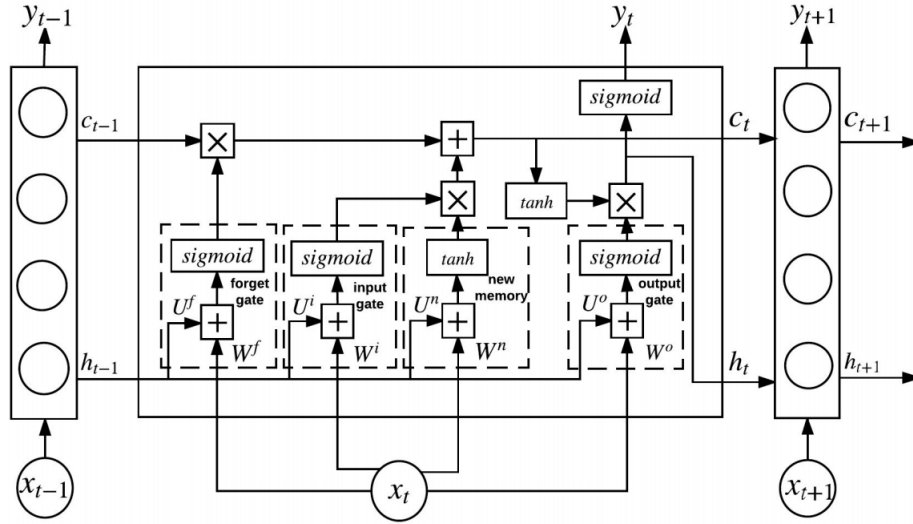


Fig. 2.15. Red de gran memoria a corto plazo (L. Zhang et al., 2018)

De nuevo L. Zhang et al. (2018) provee una explicación sobre esta arquitectura. Si en las RNN se contaba únicamente con una capa que realmente se repite produciendo distintos estados, siendo cada uno de esos estados una capa oculta, en las redes LSTM además de ésta capa se cuenta con un bloque LSTM que contiene cuatro capas ocultas encargadas de gestionar la “memoria” de la red. Además de los estados h_t , esta arquitectura cuenta con estados de celdas de memoria. En la figura 2.15 se observa un ejemplo de arquitectura LSTM. Primeramente, se decide qué información eliminar del estado de la celda a través de una función sigmoideal llamada *forget*

*gate*⁵. Esta función toma la salida de la capa oculta anterior (h_{t-1}) y la entrada x_t y da como resultado un cero (eliminar) o un uno (mantener). Después, se escoge la información que se a mantener en el estado de la celda en dos pasos. Primero, otra función sigmoideal llamada *input gate*⁶ decide los valores que se van a actualizar y después una función arcotangente crea el nuevo vector candidato C_t que se añadirá al estado de la celda. Hecho esto, el bloque LSTM actualiza el anterior estado de la celda C_{t-1} de la siguiente forma:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t \quad (2.16)$$

Dado que la *forget gate* elimina o actualiza la “memoria”, controla el gradiente de la red, lo que permite evitar el problema del descenso por gradiente, es decir, evitar que el gradiente se desvanezca hacia el 0 o que explote hacia el infinito.

Al final del LSTM se calcula la salida en base al estado de la celda. Para ello, se aplica una función sigmoideal llamada *output gate*⁷, que decide a qué partes del estado de la celda se les da salida, y después se multiplica la salida de la sigmoideal por la arcotangente del estado de la celda.

Existen una variante de LSTM llamada *Gated Recurrent unit* (GRU) que combina las puertas *input* y *forget* en una sola puerta de actualización y junta en un solo estado el estado de la celda y el estado oculto de la red. Esta variante es más simple que el LSTM está creciendo en popularidad (L. Zhang et al., 2018).

Dentro del aprendizaje profundo existen numerosas arquitecturas además de las descritas, cómo las redes neuronales recursivas o diferentes combinaciones de las arquitecturas básicas ya explicadas. No obstante, de acuerdo al estudio citado a lo largo de esta sección, las arquitecturas empleadas en tareas de análisis de sentimiento son las ya mencionadas redes convolucionales, las redes recurrentes y su variante LSTM.

En la publicación de X. Zhang et al. (2015) se emplean redes neuronales convolucio-

⁵puerta de olvido

⁶puerta de entrada

⁷puerta de salida

nales a nivel de caracteres para clasificación de texto modelando las palabras tanto con *bag-of-words* como con *word embeddings* (*Word2Vec*). Johnson y Zhang (2014) proponen un modelo con redes convolucionales que representa las palabras a través de *bag-of-words* en una capa de convolución y además tiene en cuenta la secuencia de las palabras. Los mismos Johnson y Zhang exploran en Johnson y Zhang (2016) un modelo de LSTM con embedding de regiones. Al siguiente año, Johnson y Zhang (2017), vuelven a explorar la potencia de las CNN con un modelo de CNN piramidal con baja complejidad pero muy eficiente. En McCann, Bradbury, Xiong y Socher (2017) se diseña un modelo complejo con capas de LSTM para representar vectores de palabra de cara a traducción automática y obtiene muy buenos resultados en la clasificación. La publicación de Scott Gray y Kingma (2017) utiliza bloques LSTM muy profundos a través del aprovechamiento de kernels de GPU (Unidad de procesamiento gráfico). En la tabla 2.2 se puede observar una comparación de los resultados de estas publicaciones.

TABLA 2.2. COMPARATIVA DE MODELOS PROFUNDOS
CON WORD EMBEDDINGS

Publicación	Modelo base	Mejor resultado
X. Zhang et al. (2015)	CNN	95,6 %
Johnson y Zhang (2014)	CNN	92,3 %
Johnson y Zhang (2016)	LSTM	94,1 %
McCann et al. (2017)	LSTM	95,8 %
Scott Gray y Kingma (2017)	LSTM	95 %
Johnson y Zhang (2017)	CNN	97,4 %

La aparición de los *word embeddings* y los modelos que han derivado del mismo mediante la utilización de arquitecturas de aprendizaje profundo ha dado lugar a una gran cantidad de enfoques distintos con muy buen rendimiento que se acercan cada vez más un porcentaje de exactitud perfecto en tareas de análisis de sentimiento.

2.4. La evolución de los word embeddings

Desde los métodos tradicionales de representación de palabras en vectores (*bag-of-words*) se evolucionó hacia el uso de *word embeddings* para aprender vectores de palabras teniendo en cuenta el contexto de las mismas gracias a los métodos vistos en el apartado anterior (Word2Vec, GloVe y FastText). Éste crecimiento ha ido de la mano de los avances en el campo del aprendizaje profundo y vista la literatura, ha quedado claro que el uso de modelos de *word embeddings* y aprendizaje profundo es fundamental para la tarea de análisis de sentimiento.

Sin embargo, aunque este tipo de modelos es reciente y está en continua investigación, a partir del año 2017 y comienzos del año 2018 se han comenzado a producir enfoques que intentan ir más allá del uso de los *word embeddings*. Nuevas publicaciones tratan de producir modelos de lo que llaman “*Universal embeddings*”, es decir, basándose en la idea de transferencia de aprendizaje, obtener *embeddings* entrenados en conjuntos de datos mayúsculos de forma que puedan ser empleados universalmente en todo tipo de tareas del procesamiento del lenguaje (análisis de sentimiento o traducción automática, por ejemplo). Estos modelos se han convertido en el estado del arte por los buenos resultados que han obtenido respecto a los modelos previos.

La idea nace de la problemática de entrenar *embeddings* descrita anteriormente: requieren conjuntos de datos enormes y un tiempo de cómputo que no es asumible por las organizaciones, así como tampoco lo es el coste de las GPUs necesarias.

Por un lado, en el año 2017, surgió el modelo ***InferSent*** por parte de un grupo de investigadores sobre inteligencia artificial de Facebook (Conneau, Kiela, Schwenk, Barrault & Bordes, 2017). Su investigación trata de encontrar un modelo de representación de frases universal empleando el conjunto de datos de Stanford Natural Language Inference (SNLI). Este corpus está compuesto por más de medio millón de parejas de frases en inglés estructuradas para inferencia de lenguaje natural. Esto consiste en parejas de un texto y un texto de hipótesis en las que ambos textos tienen alguna relación, la cuál puede ser una contradicción, neutral o vinculación. La idea de este estudio es demostrar que entrenando un modelo de representación de texto en un conjunto de inferencia de lenguaje se pueden capturar características

universales.

Las arquitecturas con las que experimentan son LSTM y GRU, siendo el mejor modelo una red con LSTM bidireccional. Construyendo los *embeddings* a través del modelo que plantean y aplicándolos mediante transferencia de aprendizaje, obtienen buenos resultados (por ejemplo, 92,4 % en análisis de subjetividad de texto) en 12 tareas distintas de procesamiento del lenguaje, probando así su funcionamiento universal. Aunque estos resultados no se acerquen a los de los modelos aplicados directamente a análisis de sentimiento, la fortaleza de este modelo reside en su aplicabilidad a numerosas tareas.

Este modelo era uno de los primeros en investigar la idea de universalidad de los *embeddings* y, aunque obtuvo buenos resultados en diferentes tareas, en el año 2018 han aparecido nuevos modelos entorno al mismo enfoque que han obtenido mejores resultados y se han convertido en el estado del arte.

En enero de dicho año Howard y Ruder (2018) publicaron el modelo ***ULMFiT*** (*Universal Language Model Fine-tuning*) para clasificación de texto con el objetivo ya mencionado: un método efectivo capaz de ser aplicado a cualquier tarea de procesamiento del lenguaje. Este modelo consiste en tres etapas diferenciadas con redes profundas basándose en un modelo de LSTM. Primero, se pre-entrena el modelo de representación del lenguaje con un corpus de dominio general (en su caso, Wikitext-103) para después ajustar el modelo pre-entrenado a la tarea específica que se quiere realizar y a los datos de la misma. De esta forma, al haber pre-entrenado el modelo con un conjunto general, el aprendizaje que tiene la red sobre el mismo no se olvida y lo aplica al conjunto del problema. Finalmente, se construye un clasificador que se ajusta a la tarea específica a partir del modelo de lenguaje. En la figura 2.16 se puede ver este funcionamiento.

De cara a la evaluación del modelo, se centran en las tareas de análisis de sentimiento, clasificación de preguntas y clasificación de temas, comparándose con los modelos de Johnson y Zhang (2017) y McCann et al. (2017) vistos previamente en la tabla 2.2. El primero obtenía el mejor resultado de 97,4 % y el segundo un también muy buen resultado de 95,8 %, pero este modelo no solo consigue muy buenos resultados

universalizando para varias tareas, sino que mejora los enfoques previos de análisis de sentimiento. En el mismo conjunto de datos que McCann et al. obtenía dicho resultado, este modelo alcanza una exactitud de 96,4 %, y para el conjunto de datos que empleaban Johnson y Zhang, *ULMFiT* consigue 97,8 %. La buena “universalización” de este modelo se ve en contexto si además de centrarnos en la tarea de análisis de sentimiento se observa que, por ejemplo, en la tarea de clasificación de temas obtiene un resultado de 99,2 %.

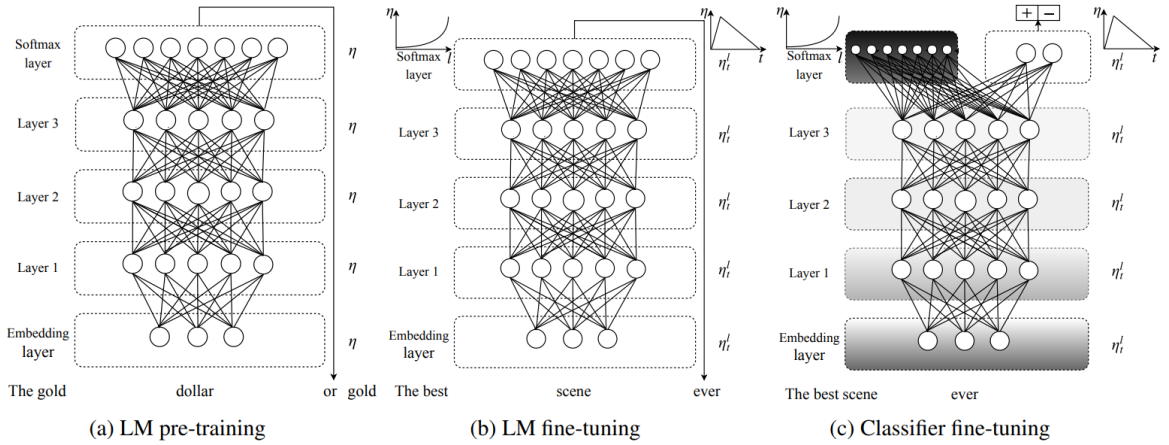


Fig. 2.16. ULMFiT (Howard & Ruder, 2018)

Este modelo se ha convertido así en el estado del arte en cuanto a clasificación de texto, pero a medida que ha avanzado 2018 han aparecido más modelos.

Otro de ellos que se ha hecho muy popular es *ELMo*, divulgado por Peters et al. (2018) con el objetivo de desarrollar representaciones de texto “profundamente contextualizadas” que mejoren los resultados del arte, pero que resulta en un buen modelo universal que logra resultados destacados en seis tareas distintas de procesamiento de lenguaje, como análisis de sentimiento o respuesta de preguntas.

De forma similar a *ULMFiT*, *ELMo* usa un modelo de lenguaje bidireccional de dos capas que se pre-entrena en un gran conjunto de datos. El modelo produce distintos *embeddings* para cada palabra dependiendo de su contexto, pues cada representación de una palabra es una función sobre la entrada. Las dos capas que emplea el modelo son bloques LSTM, aunque puede usarse con más o menos capas. Las capas superiores capturan características sobre el contexto de las palabras y las capas inferiores modelan la sintaxis.

Los investigadores no aplican este modelo a análisis de sentimiento con polaridades (positivo o negativo), sino a análisis de sentimiento multiclase (3 o 5 distintas categorías). En un conjunto de este tipo en el que el modelo de McCann et al. llega a un 53,7 % de exactitud, *ELMo* logra un resultado de 54,7 %, y respecto a su aplicación en otro tipo de tareas, el modelo consigue 92,22 % de exactitud en reconocimiento de entidades, un 0,10 % mejor que el mejor resultado previo o un 88,7 % en inferencia de lenguaje, 0,17 % más que el estado del arte anterior.

De acuerdo a los autores, *ELMo* emplea demasiado tiempo en calcular los vectores de representación, pero, como se ha visto, alcanza resultados muy buenos. A pesar de ello, no mejora los resultados de *ULMFiT*, pero el estudio de Perone, Silveira y Paula (2018) compara *ELMo* con otros ocho modelos distintos (*InferSent*, *Word2Vec*, *FastText* y *GloVe*, entre otros) aplicándolos a una variedad de tareas de procesamiento del lenguaje y del total de 33 distintas tareas, *ELMo* resulta ser el mejor modelo en 12 de ellas, mejor que el resto de modelos y solo se le acerca *InferSent*, que es el mejor en 10 tareas.

La empresa Google también ha tenido impacto en esta tendencia y dos de sus grupos de investigadores han desarrollado modelos muy destacados. El primero de ellos es ***Universal Sentence Encoder***⁸ (USE) (Cer et al., 2018), que se centra en el *embedding* no solo de palabras sino también de oraciones. El modelo plantea dos arquitecturas distintas. Por un lado, una basada en un modelo desarrollado por ellos mismos llamado “redes transformadoras”, que emplea capas de atención (capas que se centran en subconjuntos de los datos) en una composición de capas iguales llamada “codificador” y capas simples totalmente conectadas en una agrupación llamada “decodificador”. Esta vía, según los autores, es la que mejores resultados obtiene empleándose en distintas tareas de PL, pero a coste de un excesivo tiempo de cómputo y uso de memoria si la longitud de las frases es muy grande. Por otro lado, se encuentra el modelo *Deep Averaging Network*, que toma primeramente *embeddings* de palabras y bi-gramas para hacer un promedio y después pasarlo a través de una red neuronal profunda sin ninguna peculiaridad que produce el *embedding* de la frase. La ventaja de este modelo es que el tiempo de cómputo es reducido, aunque

⁸Codificador de frases universal

obtiene peores resultados que el anterior.

De nuevo, al enfocarse en lograr un modelo universal, USE se prueba con seis tareas distintas de procesamiento de lenguaje en las cuáles obtiene resultados prometedores. Sin embargo, aunque funciona bien para todas, no obtiene los mejores resultados, pues el estudio de Perone, Silveira y Paula, compara este modelo con *ELMo* y otros modelos y solo es el mejor en 8 tareas, frente a las 12 de *ELMo*. Esto no implica que sea un peor modelo, sino que en la búsqueda de la universalidad no logran los mejores resultados y por tanto, lo ideal sería emplear en cada tarea el modelo que mejor funcione.

El segundo modelo publicado por investigadores pertenecientes a Google es **BERT** (Devlin, Chang, Lee & Toutanova, 2018), el más reciente de los modelos vistos hasta ahora y quizá el más destacado, pues toma ideas de todos los modelos previos. Sus siglas significan *Bidirectional Encoder Representations from Transformers* e indican de forma aproximada el funcionamiento del modelo. El uso de un modelo bidireccional ya se había visto teniendo grandes resultados con *ELMo* y los transformadores también se aplicaban en el *Universal Sentence Encoder*.

Al igual que en el resto de estos modelos, *BERT* se focaliza en pre-entrenar el modelo a través de un conjunto de datos enorme, pero utiliza para este paso dos nuevos enfoques no supervisados.

El primer enfoque se denomina *Masked LM* (modelo de lenguaje enmascarado) y consiste en, antes de pasar las secuencias de palabras al algoritmo, enmascarar el 15 % de las palabras de cada secuencia (reemplazar la palabra por el token [MASK]). Después, las secuencias con las palabras enmascaradas se le pasan al transformado, que se encarga de predecir esas palabras partiendo de su contexto. Como de esta manera la función solo tiene en cuenta la predicción de los valores enmascarados, en cada grupo solo se predicen el 15 % de las palabras, el modelo necesita repetir este paso múltiples veces, por lo que le cuesta converger.

Y el segundo es el de *Next Sentence Prediction*, es decir, predicción de la siguiente frase, basado en la idea de tomar pares de frases y que el modelo aprenda a predecir si la segunda frase es la continuación de la primera. Esto es así pues de acuerdo a

los autores en el 50 % de los casos de entrenamiento, la segunda frase es la frase que sigue a la primera. El planteamiento asume que el modelo aprenderá los casos en los que esto no ocurre y separará ambas frases. Para que logre su objetivo, antes del entrenamiento se introducen tokens como en el enfoque anterior, pero en este caso se indica con [CLS] el comienzo de cada primera frase y con [SEP] el final de cada frase.

Ambas estrategias se emplean a la hora de entrenar un modelo de *BERT* para minimizar la función de coste del modelo y se pueden ver representadas en la siguiente figura:

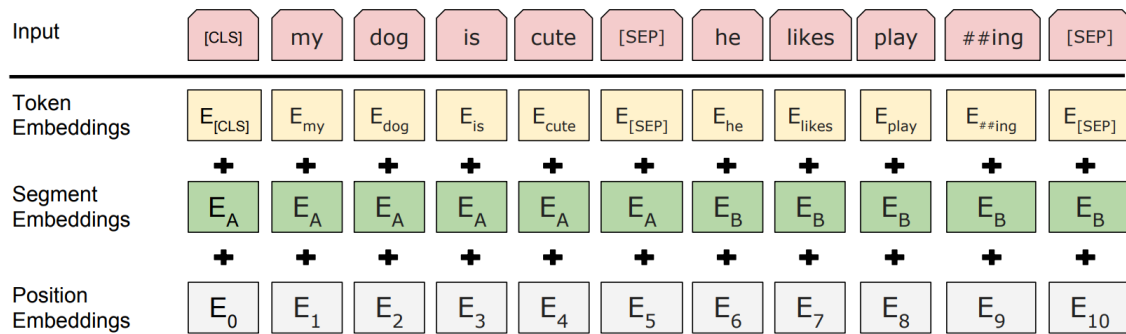


Fig. 2.17. Modelo BERT (Devlin et al., 2018)

A la hora de ajustar el modelo pre-entrenado a una tarea de clasificación, *BERT* es bastante simple y consiste únicamente en añadir capas de clasificación a la salida del transformador. Los autores evalúan los resultados del modelo en diferentes tareas para conocer cómo de universal es el modelo. Estas tareas incluyen reconocimiento de entidades, análisis de pregunta y respuesta e inferencia del lenguaje, entre otras, y el modelo demuestra los mejores resultados hasta la fecha, superando al resto de modelos, entre ellos *ELMo*. Se debe tener en cuenta que este modelo es muy reciente (octubre de 2018), por lo que en el futuro tendrá más avances. Los propios autores concluyen que aunque obtiene resultados consistentes, en algunos casos superando a los humanos, el modelo se debe seguir investigando en el futuro.

Por desgracia, dado lo reciente del modelo, no ha sido experimentado para la tarea de análisis de sentimiento ni se tienen implementaciones del modelo, por lo que en este trabajo no contamos con una base empírica de la que partir para emplear

BERT.

Existen otros modelos pero estos son los principales y más prometedores. Siendo tan recientes, este enfoque al campo del procesamiento del lenguaje continua investigándose, por lo que en el futuro existirán otros modelos alternativos a los ya mencionados. A la hora de aplicarlos, todos los modelos vistos podrían resolver eficientemente nuestro problema pero en este trabajo no se pueden aplicar todos, así que se debe escoger uno con el objetivo de experimentar con este tipo de enfoques. Considerando todo lo comentado anteriormente, se ha decidido la utilización de *ULMFiT* debido a que es un modelo que, aún estando centrado en conseguir *embeddings* universales, se ha enfocado en la clasificación de texto y es el estado del arte actual, el modelo que mejores resultados logra. Además, los autores del mismo han creado una librería de aprendizaje profundo que cuenta con la implementación de este modelo.

2.5. Clasificación multiclase

Hasta ahora, a lo largo de todo el análisis de la literatura se han comentado modelos enfocados en la clasificación del sentimiento de texto. Esta clasificación se basó inicialmente en determinar la polaridad de un texto, es decir, determinar si un texto es positivo o negativo. En este análisis, se ha visto como evolucionaban las formas de representar el texto para poder clasificarlo, pero también han evolucionado los intereses a clasificar. Entre estos nuevos intereses, como por ejemplo la clasificación de la determinada emoción que representa un texto, se encuentra la clasificación de grano fino (o multiclase), el nombre que recibe el tipo de clasificación que trata de clasificar el texto no de forma binaria, sino en cinco categorías (muy negativo, negativo, neutro, positivo y muy positivo). La razón de que surgiese es muy simple, la sociedad ha producido nuevos sistemas de opiniones, recomendaciones o críticas y en ellos ha resultado más útil e intuitivo el hecho de calificar mediante estas valoraciones en lugar de simplemente ser positivo o negativo, gracias a que aporta más matices a la calificación.

La tarea de clasificar un texto en múltiples clases es más complicada que la de una

simple polaridad porque el modelo empleado debería encontrar dentro del lenguaje características determinantes que deberían diferenciar unas categorías de otras. Los propios humanos en ocasiones no somos consistentes con las diferencias entre nuestras valoraciones y, por ejemplo, lo que una persona considera como una opinión neutra, otra persona lo podría considerar un poco negativo. Al tratar con un conjunto de datos de opiniones escritas por humanos, nos enfrentamos a este problema y es posible que un modelo pueda abstraer correctamente estas características diferenciales y obtener buena exactitud. La necesidad de encontrar características más determinantes es la que da nombre a la clasificación de grano fino, por tener que inferir más minuciosamente.

El conjunto de datos con el que se cuenta para este problema pertenece a una aplicación de reseñas en la cuál las mismas se valoran con de 1 a 5 estrellas, por lo que resulta necesario revisar los resultados en este tipo de clasificación. Aunque los modelos vistos se hayan considerado en base a su clasificación de polaridad, muchos de ellos también han realizado clasificación multiclase, por lo que sirven para tener en cuenta los resultados de la literatura.

A continuación, se puede encontrar una tabla que representa la exactitud obtenida por los modelos vistos en su aplicación a un problema de clasificación multiclase.

TABLA 2.3. COMPARATIVA DE RESULTADOS EN
CLASIFICACIÓN MULTICLASE

Publicación	Mejor resultado
X. Zhang et al. (2015)	62 %
Johnson y Zhang (2016)	67,6 %
McCann et al. (2017)	53,7 %
Johnson y Zhang (2017)	69,4 %
Howard y Ruder (2018)	70 %
Peters et al. (2018)	54,7 %

Como se puede observar, la exactitud en este tipo de clasificación es bastante menor que en clasificación de polaridad/clasificación binaria dada la complejidad mayor

que tiene este tipo de clasificación. El modelo que mejor exactitud consigue es el mismo que en el caso de la clasificación binaria, *ULMFiT*, pero vistos los resultados queda claro que queda mucho por investigar de cara a lograr mejores resultados en esta tarea.

3. ANÁLISIS Y DESARROLLO DEL PROBLEMA

3.1. Planteamiento

Como se ha descrito previamente, este trabajo consiste en la investigación de los distintos modelos existentes en el campo del aprendizaje automático en busca del modelo que mejor exactitud obtenga a la hora de clasificar el sentimiento de una serie de reseñas publicadas en una red social. Desarrollado en el capítulo anterior el estado del arte actual para este tipo de problemas, el núcleo del trabajo reside en la experimentación con los modelos detallados con el fin de determinar el modelo más apropiado para esta tarea. Concretamente, el conjunto de datos de la red social Yelp contiene, entre otra información, las reseñas y su valoración correspondiente en estrellas (de 1 a 5 estrellas), por lo que se tratará de encontrar un modelo que aprenda correctamente los aspectos significativos de cada reseña y sea capaz de predecir las estrellas correspondientes. También se buscará un modelo que trate de clasificar estas reseñas según su polaridad, positiva o negativa.

En este contexto, se emplearán tres tipos distintos de modelos para clasificar las reseñas de nuestro problema: el modelo *bag-of-words* (con los algoritmos Naïve Bayes, SVM, regresión logística y Random Forest), los word embeddings (entrenados con *Word2Vec*) y los word embeddings universales (con *ULMFiT*).

Los modelos del estado del arte describen los resultados de su estudio, que son dependientes tanto de las implementaciones realizadas como de los conjuntos de datos empleados, por lo que este trabajo tratará también de obtener resultados cercanos a los de los modelos referentes. Para ello, primeramente se realizará una experimentación con los distintos modelos planteados. Cada modelo de clasificación tiene sus propios parámetros que afectan al aprendizaje del mismo, por lo que en dicha experimentación se alterarán de forma estudiada para conocer de qué forma afecta a los resultados que se obtienen.

El tratamiento de los datos y la implementación de los modelos estará realizada

completamente en el lenguaje de programación Python, dado que es uno de los lenguajes con más desarrollo dentro del campo del aprendizaje automático, gracias a la aportación de librerías como *scikit-learn* (Pedregosa et al., 2011).

Antes de comenzar la experimentación, se trazará un análisis del conjunto de datos del problema para obtener información sobre el mismo que permita emplear un enfoque correcto. Después de ella, se procederá a la evaluación de los modelos mediante métricas de exactitud, exhaustividad, rendimiento y error.

El trabajo se ajustará debidamente a las regulaciones y normativas técnicas aplicables a este contexto, las cuáles serán descritas más adelante.

Asimismo, es necesario tener presentes los trabajos similares o alternativas que se han realizado previamente con este mismo objetivo o similar. Aunque en el capítulo previo ya se han descrito estudios sobre análisis de sentimiento, se deben considerar los enfoques que se han realizado para la clasificación de reseñas en redes sociales. Sin embargo, en ciertas ocasiones, los propios estudios de los modelos de análisis de sentimiento se han basado precisamente en reseñas.

Es el caso del modelo *ULMFiT*, el cuál hemos descrito como el que mejor resultados obtiene y en cuyo estudio plantean un modelo de clasificación experimentando con seis distintos conjuntos de datos. En concreto, emplean un conjunto de datos de reseñas de películas de la página web IMDB, un conjunto de datos preparado para tratamiento de texto, un corpus que contiene texto de artículos de noticias, un conjunto de datos de texto de la DBpedia, un proyecto que recopila información de forma estructurada, y por último utiliza el conjunto de datos de Yelp, el cuál empleamos nosotros, en dos variantes: clasificación multiclase y clasificación binaria. Aunque el que mejor resultados da es el conjunto de la DBpedia, el que está compuesto por reseñas de redes sociales, es decir, el de Yelp, logra una exactitud del 97,84 % en clasificación binaria y 70,02 % con 5 clases (Howard & Ruder, 2018).

Lo mismo ocurre con Johnson y Zhang (2016) y Johnson y Zhang (2017), que aunque no plantean una solución para un problema de clasificación de reseñas, emplean el *dataset* de Yelp para experimentar en busca de un modelo para clasificación de texto. Con él, obtienen un buen resultado de 69,42 % con todas las categorías y un

97,36 % en polaridad. Al mismo tiempo, emplean otros conjuntos de datos, como los mencionados en el párrafo anterior o, por ejemplo, el corpus de Yahoo, que contiene preguntas y respuestas de su web Yahoo! Respuestas. De nuevo, el corpus utilizado que mejores resultados da es el de la DBpedia y el sentido de esto es que, en el caso de Yelp, el vocabulario que se emplea en las reseñas es específico de ese contexto, mientras que en la DBpedia se encuentran datos de todo tipo. Al emplear vocabulario general, los modelos son capaces de abstraer mejor las características del texto.

X. Zhang et al., 2015 es otro caso igual a los anteriores: emplea distintos conjuntos de datos, entre ellos los de Yelp, y obtiene mejores resultados con el corpus de DBpedia. Este estudio es el primero en el cuál se utiliza el conjunto de datos de Yelp, pues el año de publicación del mismo es el año en el que se hizo público por primera vez este corpus.

Las comparaciones entre estos enfoques ya han sido realizadas en las tablas 2.3 y 2.2, en el capítulo anterior.

El hecho de que otro conjunto de datos proporcione mejores resultados que el de Yelp para clasificación de texto no nos es relevante, pues el foco de este trabajo está en la clasificación de reseñas en redes sociales y el conjunto escogido es el de Yelp.

La existencia de diversos estudios enfocados a la clasificación de reseñas en redes sociales viene dada por la convocatoria anual de *SemEval*, un taller en el que se convoca a los investigadores a desarrollar modelos de clasificación de sentimiento que serán evaluados de acuerdo a una serie de tareas. En el año 2017, uno de los conjuntos de tareas estuvo centrado específicamente en la clasificación de sentimiento en redes sociales (Twitter, por ser la más utilizada). En Nakov, Ritter, Rosenthal, Sebastiani y Stoyanov (2016) se encuentran desarrollados los resultados para la tarea específica de clasificación de sentimiento de los *tweets* y la subtarea A trata particularmente de la clasificación de la polaridad.

El estudio de Guzman y Maalej (2014) es muy similar a éste: aplica las distintas técnicas existentes para clasificar el sentimiento de una serie de reseñas. La principal diferencia es que en nuestro caso el contexto es el de la reseñas de hostelería y en el

suyo se trata de reseñas de aplicaciones móviles. Lamentablemente, al publicarse en el año 2014, todavía no se habían desarrollado los que actualmente son los modelos del estado del arte, por lo que emplea modelos distintos a los descritos en este documento.

L. Zhang et al. (2018) desarrollan en su publicación una visión general de los modelos existentes en clasificación de sentimiento, algo similar a uno de los objetivos de este trabajo.

Una de las páginas web más importantes (competidora de Yelp) en lo relativo a reseñas de hostelería y servicios es TripAdvisor. Otro sitio web en el que se encuentran reseñas de productos es Amazon, que incluye reseñas de 5 estrellas sobre los productos vendidos en la tienda. No obstante, vistos los modelos del estado del arte, en ninguno de ellos se ha empleado un conjunto de reseñas de estas páginas para la experimentación. El motivo de ello es que estas empresas no han favorecido el uso de sus datos distribuyendo los mismos públicamente y de forma estructurada, sino que para su uso es necesaria la aplicación de técnicas de *web scraping*. A pesar de ello, se puede encontrar artículos en la web que aplican este tipo de técnicas, como el artículo de Li (2018), que obtiene todas las reseñas existentes en TripAdvisor correspondientes a un hotel y aplica análisis de sentimiento para clasificarlas.

Como se ha comentado previamente, varias de las publicaciones también empleaban clasificación de reseñas pero, en lugar de hostelería y servicios, de películas, con el conjunto de datos de IMDB. Este es un problema similar al nuestro y también ha sido explorado en el sitio web Rotten Tomatoes, que al igual que IMDB, está dedicado a las reseñas de películas. En concreto, la publicación de los investigadores de Stanford Socher et al. (2013), cuyo modelo ha tenido mucha relevancia en el área (más de 2400 referencias según Google Scholar), empleaba un conjunto de datos de dicha página.

Además de los mencionados, existen numerosos textos publicados explorando el análisis de sentimiento mediante otros procedimientos y utilizando otros dominios diferentes.

Realmente, los trabajos similares que se han podido encontrar difieren de éste en

que se centran en la explotación de un mismo modelo sobre un conjunto de datos o analizan los resultados de los distintos modelos en base a los resultados publicados por los propios autores, mientras que en este trabajo se van a aplicar los distintos modelos en busca de uno que resuelva el problema planteado.

3.2. Metodología

A la hora de llevar a cabo la experimentación se desarrolla una metodología, un flujo de trabajo a seguir para construir de forma estructurada los modelos que se quieren emplear para el análisis del sentimiento. De esta forma, cada paso se puede entender como un componente distinto que tiene una funcionalidad determinada dentro del problema, dónde la entrada de uno depende de la salida del anterior, siendo la entrada del flujo el conjunto de datos y la salida el modelo resultado.

Llevar a cabo una metodología permite planificar y estimar el trabajo a realizar, preparar un plan de desarrollo y centrar el foco en cada una de las fases de forma independiente. En la figura 3.1 se puede ver representado el procedimiento.

En los problemas de aprendizaje automático no es posible conocer qué modelo o algoritmo da el mejor resultado sobre un conjunto de datos sin haber experimentado previamente, por lo que a la hora de escoger un modelo para un determinado problema lo único que se puede hacer es prueba y error, es decir, probar con distintas representaciones del conjunto de datos, distintos algoritmos y distintos parámetros de cada algoritmo, motivo por el cuál se debe seguir un procedimiento.

Ya se ha explicado anteriormente que el problema de la clasificación de texto mediante aprendizaje automático es el de representar el texto de forma que un algoritmo lo pueda interpretar, y en todos los modelos de aprendizaje automático una de las tareas principales previas a la experimentación es la preparación de los datos, pero en este caso esta tarea es aún más importante, pues el texto cuenta con elementos como signos de puntuación, mayúsculas y minúsculas o palabras que se repiten, y estos elementos no son relevantes para los algoritmos, por lo que se debe procesar el texto.

1. Obtención de datos: en esta fase se reúnen los datos necesarios para el problema a partir de un origen de datos fiable (Yelp, en este caso) y se realiza un análisis y una exploración los mismos, pues es necesario interpretar de forma precisa los datos para plantear su procesamiento y tener clara una estrategia de experimentación. La importancia de esta fase reside en que la calidad y la cantidad de los datos obtenidos definirá el rendimiento del modelo. Ésta fase se desarrolla en la siguiente sección.

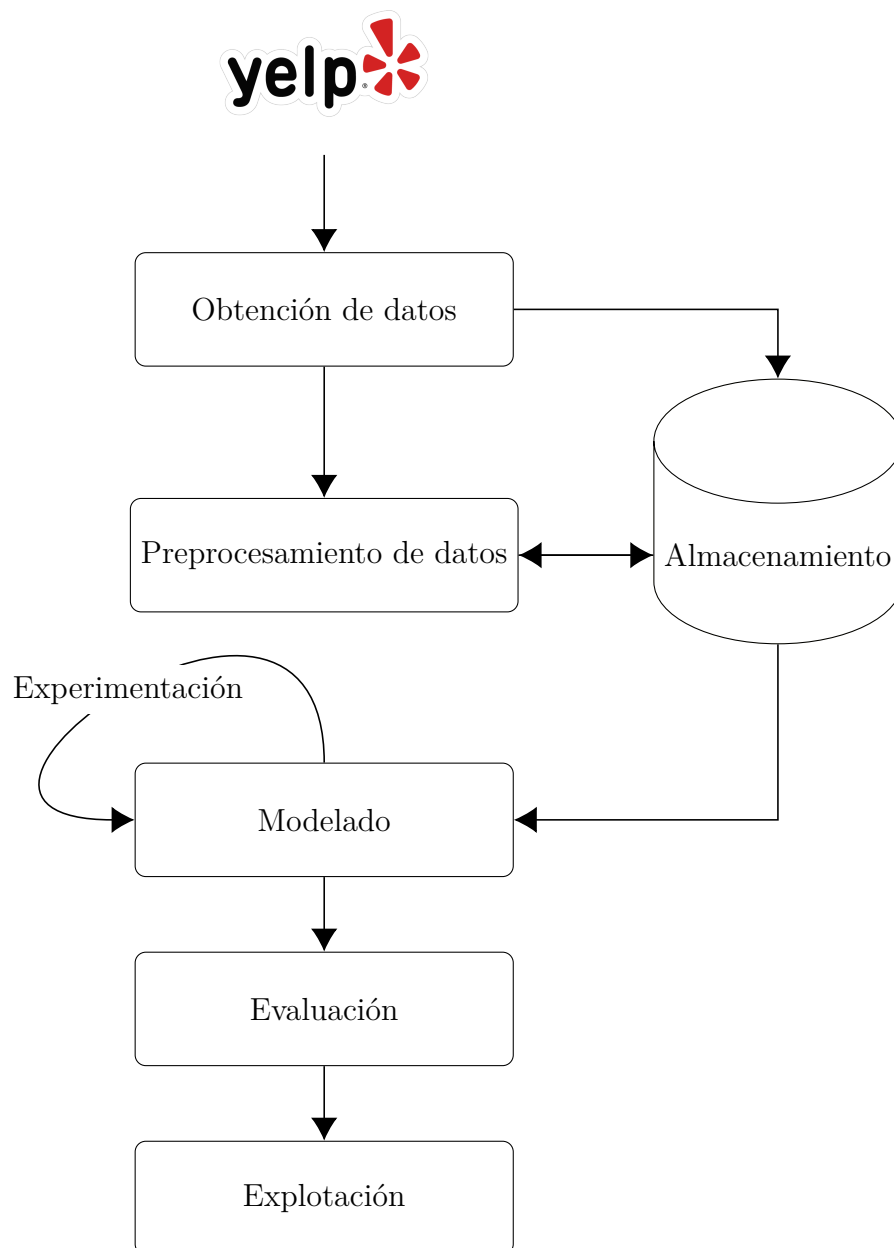


Fig. 3.1. Metodología de trabajo

2. Preprocesamiento de datos: cuando ya se cuenta con los datos que se van a clasificar, en concreto con el texto en este problema, se debe limpiar y preparar para que un algoritmo de aprendizaje automático lo pueda interpretar. La correcta preparación de los datos es vital para el rendimiento de un modelo de clasificación de texto. Esta etapa de la metodología se describe en la sección 3.3.2.

Paralelamente a las tareas de obtención de datos y preprocesamiento se encuentra una tarea de almacenamiento. La razón de esto es que es necesario contar con el conjunto de datos de forma organizada y de fácil acceso, por lo que se almacenará en una base de datos de la cuál se obtendrá lo necesario para el procesamiento, cuyo resultado también se guardará de cara a su uso en la experimentación.

3. Modelado: la etapa de modelado consiste en, una vez se tienen los datos preparados para su tratamiento, emplear las distintas implementaciones de los modelos a probar, variando los parámetros de los mismos en busca de distintos resultados hasta llegar al mejor resultado final. Dentro de esta etapa se ubica la carga principal del aprendizaje automático: el entrenamiento de los modelos y su validación.
4. Evaluación: la experimentación con distintos modelos da resultado a una variedad de modelos con distintos parámetros y características que funciona de distintas maneras. El objetivo de la fase de evaluación es utilizar distintas métricas (como la exactitud) para determinar qué modelos son los mejores. Esta fase va unida a la experimentación (modelado).
5. Explotación: terminado el desarrollo de un modelo de clasificación, habiéndose entrenado y validado, su ciclo de vida continúa para emplearlo la predicción de reseñas, esto es, para usarlo en su objetivo final.

Esta metodología está inspirada en el modelo CRISP-DM (Wirth, 1999), una metodología de referencia para proyectos de minería de datos que se acopla muy bien a proyectos de aprendizaje automático, pues al fin y al cabo son proyectos complementarios.

3.3. El conjunto de datos

La compañía Yelp, encargada del desarrollo del sitio Yelp, aplicación de búsqueda de servicios con reseñas, publica semestralmente y de forma libre un subconjunto de los datos de la aplicación con fines académicos y y de investigación. Junto a esta publicación, lanzan lo que llaman el “Yelp Dataset Challenge”, en el que retan a cualquier estudiante o investigador a tratar de explorar las posibilidades que ofrece el dataset y realizar análisis e investigaciones sobre él.

Lo que hace a este conjunto propicio para muchas tareas de investigación es que contiene 5261669 reseñas sobre 174657 establecimientos tales como restaurantes, hoteles o gimnasios, además de casi 300000 fotografías sobre los mismos. Asimismo, tiene información adicional de los establecimientos, como consejos de los usuarios, disponibilidad, ambiente, etcétera.

En un principio, buscando un objeto de estudio partiendo de este conjunto de datos, se vio que se podían aplicar numerosas técnicas, como por ejemplo reconocimiento de imágenes o análisis de grandes volúmenes de datos, pero teniendo en cuenta la gran cantidad de reseñas que ofrecía y el estado del arte del procesamiento del lenguaje (ver Capítulo 2), se decidió aplicar análisis de sentimiento.

El conjunto de datos en cuestión se ofrece en formato JSON y contiene los siguientes ficheros, que corresponderían a las tablas de la base de datos de Yelp:

- `business.json`: almacena información acerca de los negocios, como su localización, su categoría o sus atributos.
- `review.json`: contiene el texto completo de las reseñas, además del usuario que la escribió y el negocio al que se refiere.
- `user.json`: datos de los usuarios, incluyendo los metadatos asociados al usuario y la información sobre sus amigos.
- `checkin.json`: checkins de los usuarios en un negocio.
- `tip.json`: consejos escritos por los usuarios acerca de un negocio. Son más cortas

que las reseñas y son sugerencias rápidas.

- photo.json: contiene datos sobre las fotografías, incluyendo el título de la misma y su clasificación.

Considerando todos los datos con los que cuenta el dataset, se puede apreciar la cantidad de posibilidades que existen para estudiarlo. No obstante, de cara a nuestro problema de análisis de sentimiento, el único subconjunto que nos interesa es el de las reseñas, pues es el que contiene el texto de las mismas y las estrellas que les corresponden. Por tanto, únicamente se dedicará tiempo a analizar éste.

Cabe destacar que, al ser una aplicación de mayor popularidad en Estados Unidos, el idioma de la gran mayoría de las reseñas es el inglés, por lo que se han tomado las reseñas de este idioma como principales. Realmente, esto no es una decisión del trabajo, sino que al ser mayoritarias, el modelo que se desarrolle muy posiblemente se ajustaría a las palabras en inglés, por lo que el estudio se va a centrar en éstas.

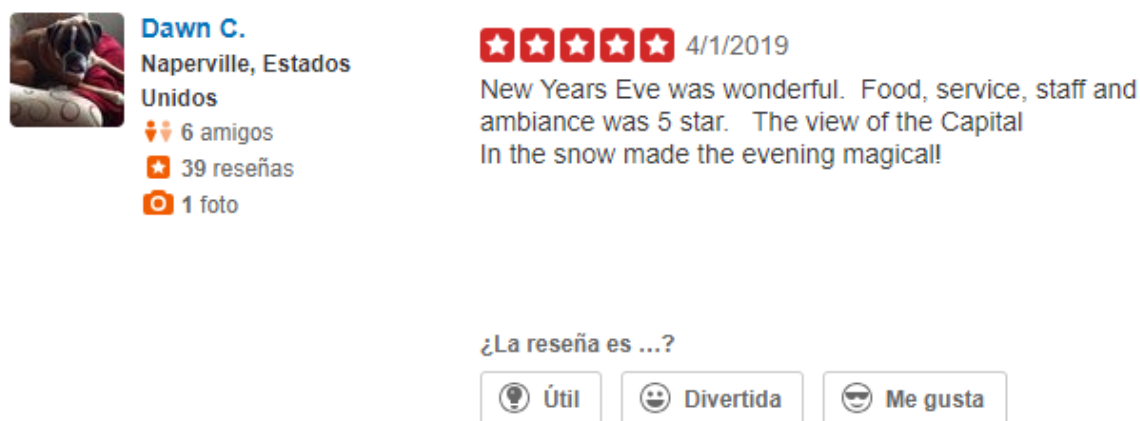


Fig. 3.2. Ejemplo de reseña en Yelp

Las reseñas, además del texto y la valoración, incluyen otra información, como una serie de votos que los usuarios pueden dar a cada una de las reseñas. En este problema únicamente nos interesa el texto de la reseña (como entrada del modelo) y las estrellas que se han valorado (como variable clasificatoria en el modelo), aunque en otros estudios sería interesante explorar la relación que tienen otras variables del conjunto de datos total. Los campos de las reviews son los descritos en la tabla 3.1.

TABLA 3.1. CAMPOS DE DATOS DE REVIEW

Campo	Descripción
review_id	Identificador único de la reseña
user_id	Identificador único del usuario que ha enviado la reseña
business_id	Identificador único del negocio al que refiere la reseña
stars	Valoración en estrellas
date	Fecha de la reseña
text	La propia reseña
useful	Número de votos de “Útil” recibidos por la reseña
funny	Número de votos de “Divertida” recibidos por la reseña
cool	Número de votos de “Me gusta” recibidos por la reseña

Esta tabla representa al mismo tiempo el modelo de datos del problema, pues en nuestra base de datos se cuenta únicamente con la tabla *review*.

Una vez se tiene en mente los datos que nos interesan, se puede pasar a analizarlos para entenderlos mejor antes de procesarlos y pasar a la experimentación.

	stars
count	5261669
mean	3.727740
std	1.433593
min	1.000000
25%	3.000000
50%	4.000000
75%	5.000000
max	5.000000

Fig. 3.3. Estadísticas descriptivas de las valoraciones

Si se observa la figura 3.3. se pueden apreciar las estadísticas descriptivas de la

tendencia de las valoraciones. La más importante de todas sería la media (*mean*), que nos permite saber que la tendencia del conjunto de reseñas es positiva, aunque lo apropiado es conocer la distribución según cada número de estrellas. El máximo y el mínimo de los valores, o el número de reseñas no son datos que no conociésemos ya, pero la desviación estándar (*std*) resulta interesante. Ésta cuantifica la dispersión de los datos respecto a la media, por lo que una desviación estándar de 1,434 indica que la mayor parte de los datos se encuentran dentro del intervalo $[2,29, 5,16]$. Si se ajusta el mínimo y el máximo de este intervalo, teniendo en cuenta que las estrellas son una variable discreta (pueden ser 1, 2, 3, 4 o 5, pero en ningún caso tomar un valor intermedio), quedaría que la mayor parte de los datos se encuentran en 3, 4 o 5 estrellas. Esta distribución se puede ver representada en la siguiente gráfica:

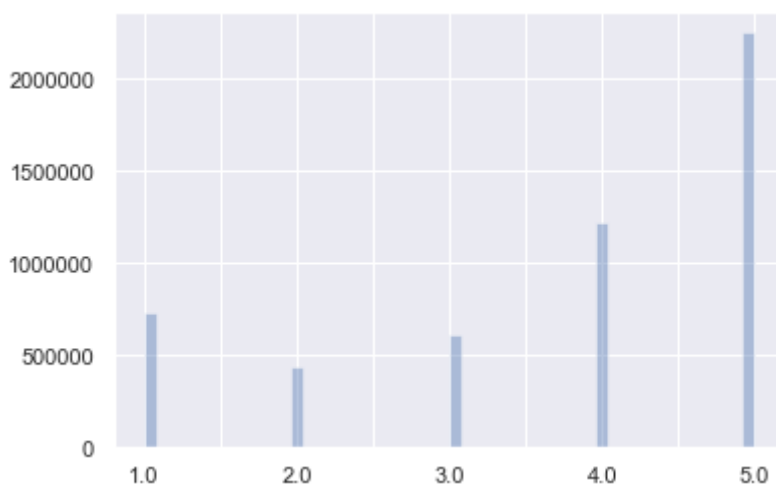


Fig. 3.4. Distribución de las reseñas

Por otro lado, los percentiles nos indican que el 25 % de la muestra se encuentra por debajo de 3 estrellas y que el 50 % del conjunto está compuesto de reseñas valoradas con 4 o 5 estrellas, lo que nos reafirma lo que se ha visto previamente: la tendencia de los datos es a contener reseñas positivas, valoradas con más de 3 estrellas.

La desventaja de haber identificado esta tendencia es que puede existir un sesgo a la hora de entrenar el modelo, debido a que no existen tantas muestras de estrellas positivas como negativas. Como se ve en la gráfica, la diferencia es muy significativa: reseñas de 2 estrellas hay menos de 500,000, mientras que existen más de 2,000,000 de reseñas con 5 estrellas. Esto es algo que se debe solucionar dado que el hecho

de que exista sesgo puede evitar que nuestro modelo detecte relaciones relevantes entre las variables, por lo que en la sección de preparación de datos describiremos las medidas tomadas para paliar esta tendencia.

Se podría formular alguna teoría que explicase el motivo de que los usuarios publiquen mayoritariamente reseñas positivas pero sería necesario algún tipo de estudio empírico, posiblemente sociológico, que lo demostrase. Sin ningún estudio de por medio, podríamos plantear que quizá se deba a que simplemente existen más usuarios a los que les ha gustado el restaurante al que han ido que usuarios a disgusto, o que quizá a los usuarios les guste más compartir su opinión cuando han tenido una buena experiencia, pero ambos planteamientos carecen de base científica.

Hasta ahora, se han estudiado las valoraciones respecto a su número de estrellas, pero se va a intentar relacionar, mediante la observación, el número de estrellas con el texto, adelantando lo que debería conseguir el modelo entrenado de forma correcta.

Un factor que podría tener algún tipo de correlación con el número de estrellas es la longitud del texto. Si visualizamos cada número de estrellas en función de la longitud (en número de palabras) de sus reseñas:

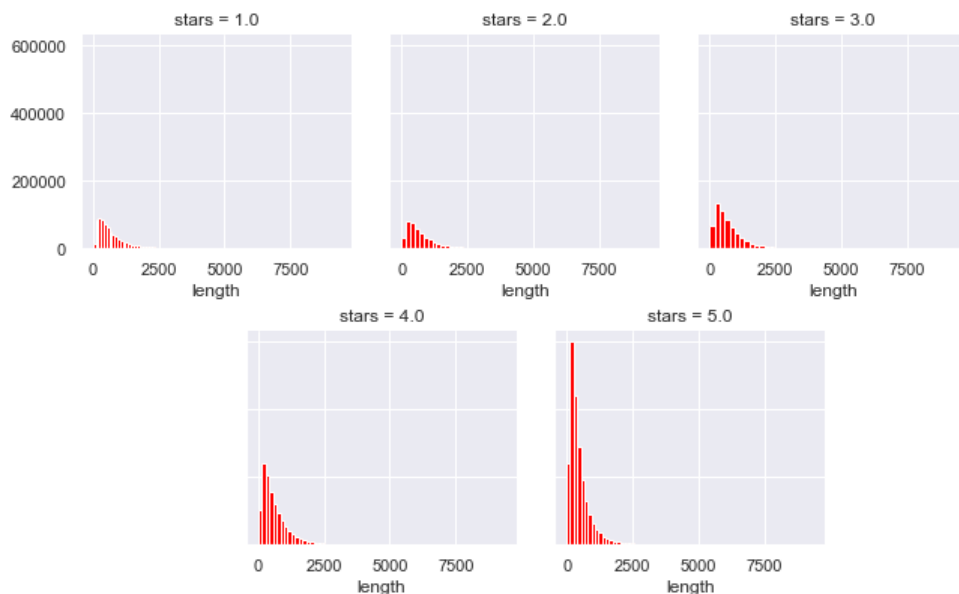


Fig. 3.5. Longitud de las reseñas en función de sus estrellas

A priori, la longitud de las reseñas parece similar para los distintos números de estrellas, pero si vemos la media de la longitud para cada número de estrellas (tabla 3.2), resulta que las reseñas negativas tienen, de media, mayor longitud que las reseñas positivas. Se podría pensar que el motivo de esto es que quizá los usuarios más disgustados dediquen más tiempo a escribir su reseña.

TABLA 3.2. MEDIA DE LONGITUD DE LA RESEÑA EN
FUNCIÓN DE LAS ESTRELLAS

Número de estrellas	Media de longitud
1	766.36
2	772.29
3	719.57
4	634.39
5	494.55

La longitud de las reseñas es mayor en las negativas que en las positivas, aunque la diferencia no es tan significativa como en el número de reseñas por estrella, por lo que existe una pequeña correlación entre la longitud y el número de estrellas. Como ya se ha mencionado antes, el sesgo existente al haber distinto número de reseñas por estrella se podría arreglar normalizando el conjunto de datos, pero en el caso de que la longitud sea distinta según el número de estrellas es otra desviación importante que es más complicado solucionar. La razón de que esto sea un sesgo es que las reseñas con mayor longitud contienen mayor número de palabras que el modelo procesará, ajustándose de distinta manera a estas que a aquellas con menor número.

Observando los datos, la diferencia, aunque la hay, no es mucha, por lo que el impacto en el problema no debería ser alto. Aún así, se ha intentado buscar una solución a este problema, dado que no podemos saber con certeza a priori si esto afectaría al problema, pero no se ha encontrado ninguna solución. Una podría ser normalizar la longitud de las reseñas, pero la pérdida de las palabras de aquellas reseñas que se acortasen significaría una pérdida del contexto, lo cuál podría suponer un problema aún mayor.

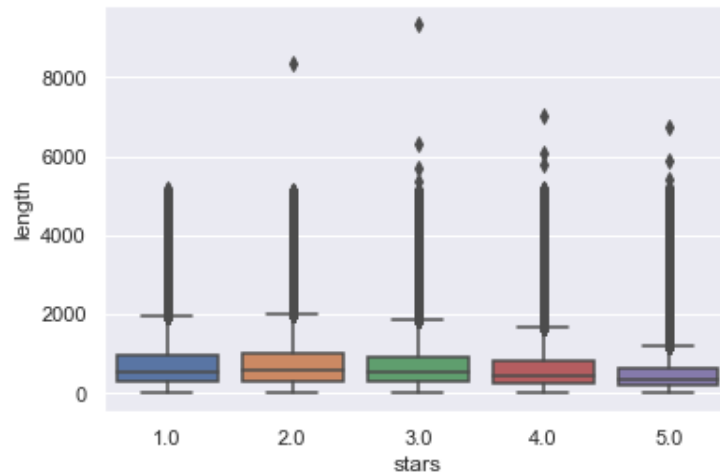


Fig. 3.6. Diagrama de cajas de longitud de la reseña en función de las estrellas

Si representamos la longitud y las estrellas en un diagrama de cajas, de nuevo queda claro que las reseñas de 1 y 2 estrellas son más largas que el resto, pero en el diagrama podemos ver que hay mucha presencia de valores atípicos, por lo que se concluye que valorar este factor no es interesante para el problema. También se puede ver a través de este diagrama que la distribución de los datos de cada estrella no es simétrica (aunque este dato tampoco es relevante).

Después de esto, se considera que lo que queda por analizar son las palabras del texto, las cuáles serán analizadas y procesadas en las siguientes secciones.

3.3.1. Obtención de los datos

El conjunto de datos que se ha descrito se puede descargar de forma libre desde el sitio web de Yelp (<https://www.yelp.com/dataset>). El fichero comprimido pesa más de 2,5 gigabytes de memoria y descomprimido más de 6,3 gigabytes, conteniendo los ficheros en formato JSON descritos en el apartado anterior.

El formato que se va a emplear para representar los datos y poder utilizarlos en las implementaciones de los modelos es CSV (*comma-separated values*), por lo que es necesario transformar los ficheros descargados de formato JSON a CSV, lo cuál se puede realizar gracias a que junto con el conjunto de datos, Yelp provee un ejecutable

para convertir los ficheros de JSON a CSV. Python cuenta con la librería *pandas* (McKinney, 2010) para manejar estructuras de datos en formato CSV, la cuál se va a utilizar para trabajar con el conjunto de datos. Esta librería gestiona los datos a través de una estructura tabular llamada *DataFrame*.

El conjunto se almacena en una base de datos para tener fácil acceso al mismo. En concreto, se utiliza el gestor de bases de datos MySQL por estar extendido y tener una licencia de uso libre. Utilizando el paquete *SQLAlchemy* de Python se puede abrir una conexión con la instancia del gestor e importar todos los datos del conjunto gracias a *pandas*, que contiene una función *to_sql* que permite escribir los datos de un *DataFrame* en una base de datos. La ventaja de este procedimiento es que la función de *pandas* deduce la estructura de los datos en SQL a partir del *DataFrame* y los importa a una base de datos de forma rápida y eficaz.

Una vez se cuenta con los datos disponibles para trabajar, es necesario reducir el conjunto a lo que realmente es necesario para el problema. Como se mencionó en la sección previa, de los ficheros .json disponibles en el dataset solo nos interesa el review.json, y dentro de éste nos centramos en el texto de las reseñas y en las estrellas valoradas, por lo que obtendremos únicamente estas columnas de la tabla de la base de datos. El conjunto de trabajo tiene la siguiente forma:

	text	stars
0	My friends and I stayed in the standard room h...	3.0
1	Super friendly staff, lowkey the best jerk you...	5.0
2	My family and I came in on Mother's Day after ...	1.0
3	My children love going to Kids Play Cafe. The ...	5.0
4	I found my Boo Boo online, went down to check ...	5.0

Fig. 3.7. Muestra del conjunto de datos

Partiendo de este conjunto, se hacen ciertas variaciones. Por un lado, no es necesario contar con el conjunto completo, dado que sus dimensiones son muy superiores a las necesarias para entrenar un modelo de aprendizaje, lo que provocaría, a priori, tiempos de cómputo excesivos sin apenas mejorar el modelo. Un conjunto de este

tamaño sería apropiado para análisis de datos masivos pero a pequeña escala, por lo que en este caso reduciremos el conjunto a 125000 reseñas, teniendo un número de reseñas más reducido que el inicial pero que al mismo tiempo es muy grande, lo que permitirá evitar el posible sesgo de que el conjunto sea demasiado pequeño y no haya suficientes ejemplos para que el modelo entrene bien.

La reducción del conjunto se hace obteniendo las reseñas de forma aleatoria, para evitar cualquier posible condicionamiento por el orden de las reseñas.

Por otro lado, se describió en el análisis previo del conjunto el problema de la diferencia de cantidad de reseñas por número de estrellas. Para solucionarlo, al mismo tiempo que reducimos el tamaño del conjunto llevamos a cabo una selección de las reseñas de acuerdo a sus estrellas, por lo que, si el objetivo es contar con 125000 ejemplos, se toman 25000 reseñas de cada tipo de estrella. De esta forma, el conjunto queda “normalizado” y el número de reseñas por estrella está balanceado, evitando la desviación existente. Aún así, podría existir el sesgo de la longitud de las reseñas, habiendo más palabras en un tipo de reseñas que en otras, pero no se ve solución a este pequeño problema y en la sección anterior ya se han mencionado razones para considerarlo “aceptable”.

En este punto entra otro factor: la clasificación binaria y la clasificación en cinco estrellas. En este problema se quiere experimentar tanto para clasificar reseñas en positivas o negativas como en sus 5 posibles estrellas, pero formando el conjunto como se acaba de mencionar habría un problema: para las 5 estrellas, existirían 25000 reseñas de cada estrella, pero en el caso de que se juntasen las dos primeras estrellas como negativas y las tres últimas estrellas como positivas, el resultado sería un conjunto descompensado de nuevo, con 50000 reseñas negativas y 75000 reseñas positivas.

Para solucionar este problema se podrían eliminar de la ecuación las reseñas con 3 estrellas, por ser la estrella “neutral”, teniendo un balance entre negativas y positivas. No obstante, valorar con 3 estrellas, y no 4 o 2, ha sido una decisión que han tomado los usuarios, existiendo un comportamiento detrás de esta decisión que el modelo debería aprender, por tanto, no parece adecuado excluir estas reseñas del problema.

Por tanto, la decisión que se ha tomado ha sido considerar las reseñas de 3 estrellas como “no negativas”, añadiendo las mismas a las reseñas de 4 y 5 estrellas como positivas y balanceando el conjunto con 62500 estrellas positivas y tantas negativas.

Así, a priori, se tendrían dos conjuntos distintos según el tipo de clasificación:

- Conjunto para clasificación binaria: este conjunto contiene reseñas clasificadas como positivas o negativas, habiendo 62500 reseñas negativas (reseñas de 1 y 2 estrellas) y 62500 reseñas positivas (3, 4 y 5 estrellas). Las reseñas negativas se etiquetan con un 0 y las positivas con un 1.
- Conjunto para clasificación multiclase: este conjunto tiene un total de 125000 reseñas, con 25000 reseñas para cada tipo de estrella (de 1 a 5).

Además de obtener el conjunto de datos para nuestro problema, se ha descrito que se emplearía el modelo *Word2Vec* para realizar transferencia de aprendizaje utilizando vectores ya entrenados. Por esta razón, es necesario también obtener dichos vectores para poder aplicarlos en este problema. Estos vectores los proporcionan los propios investigadores de Google que desarrollaron el modelo (Mikolov et al., 2013a) en (<https://code.google.com/archive/p/word2vec/>) y han sido entrenados en un vocabulario de Google Noticias que contiene cerca de 100 mil millones de palabras. El modelo que se obtiene cuenta con vectores de 300 dimensiones de 3 millones de palabras y frases. Para su uso, se incorporan los vectores pre-entrenados al modelo mediante transferencia de aprendizaje, lo cuál se describirá posteriormente.

3.3.2. Preprocesamiento de los datos

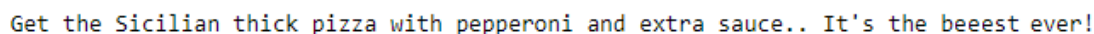
En todo desarrollo de un modelo de aprendizaje automático, antes de la experimentación con los datos de entrada, éstos se deben preparar para que el algoritmo los procese correctamente y de la forma óptima. Este proceso depende del problema y en general trata de transformar los datos para que un algoritmo los pueda entender de la forma que se quiere. Por ejemplo, en un problema con datos numéricos en el que una de las variables de entrada sea un peso (250 kg) y otra sea el precio (100 euros), las cantidades se encuentran en distintas escalas, pero los algoritmos requie-

ren que todos los datos estén en la misma escala, por lo que se normalizarían ambas variables para tener un valor entre 0 y 1.

En el caso de un problema cuya entrada es texto, esta parte es esencial: una misma palabra podría estar escrita de distintas maneras o mal escrita (por ejemplo, *hotel* y *hOTeL*), podría haber palabras que se repitan demasiado y no sean relevantes, o podría contener signos de puntuación que no sea relevante analizar, por ejemplo. Un texto bien procesado puede lograr mejores resultados que uno que no lo está.

La tarea que nos ocupa se centra en limpiar el texto del conjunto de datos para eliminar todos los aspectos irrelevantes o que no faciliten el rendimiento del modelo. Para ello, se va a hacer uso de la librería NLTK (Bird, Klein & Loper, 2009) de Python, específica para el procesamiento del lenguaje natural. Esta tarea es complicada y no es posible conocer si una modificación al texto puede afectar mejor o peor al resultado del modelo, por lo que se tratará de realizar lo más simple posible.

Con el fin de visualizar el resultado del proceso, se observará a continuación una reseña de ejemplo y al final del procesamiento se volverá a visualizar la misma reseña:



```
Get the Sicilian thick pizza with pepperoni and extra sauce.. It's the beeeest ever!
```

Fig. 3.8. Reseña no procesada

1. **Normalización:** el primer paso de este preprocesamiento consiste en normalizar el texto, es decir, que distintas representaciones de una misma palabra pasen a tener una representación común. En este caso, se va a normalizar todas las palabras a minúscula. Por ejemplo, la palabra “pizza” podría estar escrita de esta forma, pero también podría ser “Pizza”, o si alguien la ha escrito incorrectamente podría ser “piZzA”. De esta manera, se logra que todas estén representadas como “pizza”.
2. **Eliminación de caracteres repetidos:** es común en textos que no son de carácter formal (como en redes sociales) la repetición de los mismos caracteres con el objetivo de producir énfasis. Por ejemplo, en lugar de escribir “nice”, en una reseña se podría encontrar que un usuario ha escrito “niceeeee”, lo cuál

provocaría que nuestro modelo no captase que ambas palabras son la misma. En inglés no existen palabras que repitan más de 2 veces el mismo carácter seguido, por lo que corregimos toda aquella palabra en la que ocurra esta situación.

3. **Corrección de errores de ortografía:** aún eliminando caracteres repetidos, en inglés existen muchas palabras que contienen 2 veces repetidas el mismo carácter, por lo que no se pueden eliminar todos los caracteres que se repitan estas veces. Si eliminamos los que se repiten más, aún podría suceder que la palabra estuviera escrita incorrectamente con el mismo carácter repetido 2 veces. Por este motivo, se va a utilizar un corrector de ortografía que, además de solucionar este problema, corregirá las palabras mal escritas.

La corrección de ortografía de forma automática es una tarea aún de estudio en el campo del procesamiento del lenguaje, debido a su complejidad (si la palabra difiere mucho de la correcta es muy difícil corregirla), lo que hace que esta corrección no sea perfecta. Por ello, se utiliza una implementación básica de corrector ortográfico que corrija las palabras de la forma más simple posible.

4. **Eliminación de signos de puntuación y caracteres no alfanuméricos:** una vez se han normalizado todas las palabras, se pasa a eliminar el contenido del texto que no es relevante para el problema. Los signos de puntuación y demás caracteres no alfanuméricos no aportan ningún tipo de información en este problema, por lo que se procede a eliminar de las reseñas los siguientes caracteres: `!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}` .

En este caso, se entra en conflicto con las palabras compuestas, que en inglés se escriben con un guion intermedio “-”, las cuáles contienen información importante que no se debe eliminar al quitar los signos de puntuación. Del mismo modo, en inglés existen multitud de contracciones, como por ejemplo “I’ve”, que interesa mantener porque sea útiles para el modelo, por lo que se realiza un tratamiento especial a estas palabras para mantenerlas juntas.

5. **Eliminación de palabras vacías:** las palabras vacías son aquellas palabras comunes que aparecen muy frecuentemente en el texto y no tienen ningún tipo

de significado o impacto dentro de él. Éstas pueden ser artículos, preposiciones, etcétera. En inglés, por ejemplo, las más comunes serían “the” o “is”. Este tipo de palabras se deben eliminar porque no aportan nada al significado completo de la reseña y así, además, se reduce el tamaño del conjunto de datos.

La propia librería utilizada proporciona una lista de las palabras vacías más comunes en inglés, por lo que eliminamos de nuestras reseñas todas aquellas palabras que se encuentra en esta lista.

6. **Stemming:** el stemming consiste en reducir una palabra a una raíz llamada *stem* en inglés. La utilidad de esta modificación es que permite reducir el tamaño del vocabulario, identificando de la misma forma las distintas variaciones de las palabras. Por ejemplo, si es una reseña se encuentran las palabras “fishes” y “fishing”, ambas se representarían como “fish”.

No obstante, realizar este paso puede provocar que se pierda el significado de algunas palabras, pues las distintas variaciones de las mismas pueden provocar distintos significados. Además, en el caso de la experimentación con el modelo *Word2Vec*, la representación en el espacio vectorial de las palabras puede haber hecho que el modelo pre-entrenado haya aprendido las distintas variaciones morfológicas de ellas. Por estos motivos, se experimentará previamente con el conjunto limpio sin stemmatizar y con el conjunto stemmatizado para determinar cuál produce mejores resultados y emplearlo en la experimentación.

Gracias a que Yelp publica el dataset de forma estructurada, no ha sido necesaria la aplicación de tareas específicas para solucionar problemas como la ausencia de campos o la existencia de datos incompletos.

Por supuesto, el preprocesamiento del texto nunca es perfecto y siempre se puede mejorar, pero se considera que se han limpiado bien las reseñas y que profundizar más aún en el mismo alejaría el foco del objetivo del trabajo.

Viendo el resultado de aplicar estas técnicas a la reseña de ejemplo mencionada anteriormente se pueden observar los cambios que se realizan sobre los datos:

'get sicilian thick pizza pepperoni extra sauce best ever'

Fig. 3.9. Reseña procesada

Como se puede ver, en la reseña tomada se han reducido todas las palabras a minúscula, se han eliminado palabras irrelevantes como “the”, “with”, “and” e “it’s”, se han quitado signos de puntuación y se ha corregido la palabra “best”. La aplicación de estas técnicas al conjunto completo resulta en un conjunto preparado y listo para su uso en los modelos que se van a emplear.

3.4. Herramientas y tecnologías

En la siguiente sección se detallarán las herramientas, tecnologías y librerías utilizadas para el desarrollo de todas las fases de este trabajo.

3.4.1. Especificaciones del entorno

La ejecución de las experimentaciones ha sido llevada a cabo en un ordenador con los siguientes componentes:

TABLA 3.3. ESPECIFICACIONES DEL ENTORNO DE DESARROLLO

Tipo	Componente
Sistema operativo	Windows 10
Procesador	Intel Core i7-7700HQ (4 núcleos a 3.8GHz)
Memoria RAM	8GB DDR4 a 2400MHz
Almacenamiento	1 TB + 256GB SSD
Procesador de gráficos	Nvidia Geforce GTX 1050Ti 4GB DDR5

La capacidad de procesamiento del procesador permite grandes velocidades de cómputo de los modelos. Los modelos de aprendizaje profundo, que superan con creces las necesidades de cómputo de los modelos de aprendizaje automático tradicionales, re-

quieren mucho procesamiento, el cuál se realiza a través de las tarjetas gráficas. Las tarjetas gráficas de la marca NVIDIA contiene una arquitectura llamada CUDA que se aprovecha de la gran capacidad de procesamiento paralelo de este tipo de unidades para superar el rendimiento de los procesadores normales. Por ello, el uso de la tarjeta gráfica indicada ha favorecido la utilización de modelos profundos, aunque existen tarjetas gráficas de gama superior que tienen mayor capacidad de procesamiento. El uso de la tarjeta gráfica en lugar del procesador para experimentar con los modelos ha reducido en algunos caso el tiempo de entrenamiento de más de 1 hora a unos pocos minutos.

Por otro lado, aunque 8 gigabytes de RAM es una cantidad suficiente para el uso general, en ocasiones es problemática debido a la necesidad de trabajar con conjuntos de datos grandes y realizar procesamientos sobre ellos.

3.4.2. Tecnologías

Python

Python es un lenguaje de programación interpretado muy utilizado en el campo de la inteligencia artificial debido a su manejo nativo de las estructuras de datos y la gran cantidad de librerías que facilitan la realización de análisis de datos y de aprendizaje automático.

Por este motivo, se ha decidido utilizar este lenguaje para implementar los modelos mediante algunas de las librerías que ofrece.

Scikit-learn

Scikit-learn (Pedregosa et al., 2011) es una librería de Python dedicada al aprendizaje automático que simplifica la creación de modelos de aprendizaje gracias a que proporciona numerosas implementaciones de algoritmos tanto supervisados como no supervisados. Existen otras alternativas, como la librería de aprendizaje automático de Google, TensorFlow, o Theano, pero se ha decidido por Scikit-learn por su sencillez, dado que las otras librerías proporcionan los módulos para construir

modelos, pero no implementaciones de los propios modelos, lo cuál haría el proceso más complejo.

Esta librería se emplea para experimentar con los modelos de aprendizaje automático del modelo *bag-of-words*, desde la vectorización de palabras gracias a sus clases *Vectorizer* hasta el uso de los distintos algoritmos.

NLTK

NLTK (Bird et al., 2009) es un conjunto de módulos, también del lenguaje Python, dedicados al procesamiento del lenguaje humano. Provee herramientas que permiten su utilización para problemas lingüísticos, limpieza de datos, análisis sintáctico y gramatical y más tareas relativas al procesamiento de texto.

Dado que este problema tiene una gran parte de procesamiento de texto, se ha empleado esta librería que además funciona muy bien si se complementa con Scikit-learn.

Pandas

Pandas (McKinney, 2010) es una librería de Python para la manipulación y el análisis de datos. Su potencia reside en las estructuras de datos que ofrece y las herramientas para manejar tablas numéricas y aplicar operaciones sobre las mismas.

Pandas ofrece grandes ventajas en este trabajo a la hora de analizar los conjuntos de datos de Yelp y procesarlos para posteriormente emplearlos con los algoritmos.

La combinación de las tres librerías hasta ahora mencionadas resulta en el núcleo de la construcción de los modelos de este estudio: obtención de los datos, preprocesamiento de los mismos y modelado con la experimentación y su evaluación.

Keras

En el caso de la implementación de las redes neuronales con aprendizaje profundo, no es suficiente la utilización de Scikit-learn, dado que solo proporciona implementaciones de modelos no profundos. Por ello, es necesario el uso de la librería Keras (Chollet et al., 2015). Esta librería contiene, a alto nivel, implementaciones por bloques de redes neuronales, lo que permite construir cualquier tipo de las arquitecturas existentes.

En este problema concreto, su uso es necesario para la experimentación con los *word embeddings*, dado que requieren de arquitecturas de aprendizaje profundo en las cuáles incrustar los vectores ya entrenados. La razón de usar esta librería y no otras, como Pytorch, por ejemplo, es que Keras ofrece implementaciones de más alto nivel, que permiten diseñar redes neuronales profundas de manera más rápida y con menor complejidad.

fastai

La utilización de fastai viene dada por el hecho de que es la librería en la cuál se ha incluido la implementación del modelo *ULMFiT*, con el cuál se experimenta en este trabajo. Como tal, es una librería de aprendizaje profundo, pero no ofrece las posibilidades de Keras para construir modelos, sino que su objetivo es la enseñanza de los modelos neuronales. La librería está basada en módulos de otra librería de más bajo nivel, Pytorch.

L^AT_EX

L^AT_EX es una tecnología de creación de documentos de alta calidad con el objetivo de la publicación de documentos científicos. A diferencia de los procesadores de textos (el tradicional Microsoft Word), L^AT_EX funciona mediante macros que aplican cambios al documento para darle formato al compilarlo. La ventaja de esto es que permite centrar el foco en la escritura del texto en lugar de en el diseño del documento.

Se ha empleado \LaTeX para la realización de la memoria de este trabajo, pues favorece la presentación del documento, con un diseño profesional, y agiliza la escritura del mismo. No obstante, esta tecnología tiene una contraparte, y es que el aprendizaje de su uso y de sus macros requieren más tiempo que cualquier otra herramienta usual. Se contempló la alternativa tradicional, Microsoft Word, pero su única ventaja era el hecho de no ser necesario un largo tiempo de aprendizaje, dado que el resultado de \LaTeX es superior y esta última es una herramienta gratuita y de código libre, mientras que Word es de pago. Como ésta, existen otras alternativas de procesadores de texto tradicionales, en este caso libres, como Apache OpenOffice o LibreOffice, pero por las ventajas de \LaTeX se han descartado.

4. EXPERIMENTACIÓN

El propósito del siguiente capítulo es describir la experimentación a realizar con cada uno de los modelos, detallando el proceso a seguir para tratar de encontrar los mejores modelos.

El proceso de experimentación dentro del aprendizaje automático consiste en probar los diferentes parámetros de un algoritmo para encontrar el que mejor resultado proporcione. A la hora de experimentar con cualquier algoritmo de aprendizaje automático, no existe una regla de oro que indique cuáles son los mejores parámetros para cada modelo. El valor de un parámetro que funciona muy bien en un problema, en otro problema puede dar los peores resultados. Esto se debe a que el resultado es muy dependiente del conjunto de datos utilizado y, en el caso de las redes neuronales, a los múltiples ajustes que se pueden realizar sobre la red y a las distintas arquitecturas existentes.

Por este motivo, es necesario describir una serie de parámetros (distintos en cada algoritmo) con los cuáles experimentar sobre el mismo conjunto de datos para encontrar el que mejor se ajuste a nuestro problema. Por supuesto, los distintos parámetros se probarán en los dos conjuntos, binario y multiclase, pues pueden funcionar de distinta manera.

Dentro de cada conjunto de experimentos con un algoritmo, primeramente se establece un punto de partida o *baseline*. Este punto de partida sirve para realizar un primer experimento y observar cómo se comporta el modelo. A partir de este experimento, se van realizando cambios en los parámetros con el fin de ver cuáles producen mejores resultados, desestimando las tendencias negativas y probando con las positivas. No obstante, el cambio en un parámetro puede afectar a los demás parámetros, por lo que se tratará de, en la medida de lo posible, abarcar todas las posibilidades.

4.1. Experimentación previa

A priori, se cuentan con dos conjuntos de datos con diferentes categorías: binarias y de 5 estrellas (multiclase). Sin embargo existen posibilidades que podrían empeorar o mejorar el problema si se modificase el conjunto de datos y, de cara a definir la experimentación, es necesario observar de qué manera estas modificaciones afectan al rendimiento de los modelos y si es beneficioso realizarlas. Por un lado, se planteaba un conjunto de 125000 reseñas, bastante más pequeño que el conjunto inicial para reducir el tiempo necesario para entrenar los modelos, pero posiblemente si este fuera mayor se obtendría mejores resultados, por lo que se probará un conjunto de 250000 reseñas.

Por otra parte, en la sección 3.3.2 se describía el preprocesamiento que se realiza sobre los datos, y uno de los pasos es la stemmatización del texto de las reseñas. Este paso, que como ya se comentó reduce las palabras a su *stem*, reduce el tamaño del vocabulario del problema, pues palabras con el mismo *stem* se toman como una sola. Esto puede ser beneficioso, pero también puede perjudicar al modelo, pues al reducir las palabras se pueden eliminar afijos que proporcionen significado a las mismas. Además, en el caso de los *word embeddings*, al tomar vectores preentrenados, es posible que estos no hayan sido stemmatizados y que hacerlo sobre nuestro conjunto provoque que haya peores resultados. Por ello, se probará un conjunto limpio stemmatizado y un conjunto sin stemmatizar.

Para evitar una complejidad similar a la de una experimentación definitiva, en esta experimentación previa se probará únicamente con unos determinados parámetros de un algoritmo para cada conjunto y se evaluarán simplemente mediante la exactitud de los resultados. Asimismo, se realizarán las pruebas en ambos conjuntos (binario y multiclase), para comprobar que la tendencia sea la misma en ambos conjuntos.

La validación de los experimentos, es decir, la capacidad de generalización de cada uno, se medirá mediante la separación en un conjunto de entrenamiento (75 % del conjunto total) y un conjunto de validación o test (25 % del total).

En el caso del modelo *bag-of-words*, se prueba con un clasificador de Bayes Ingenuo

con la distribución multinomial. Para la vectorización de las palabras se tomará la frecuencia inversa de las ocurrencias y se emplearán unigramas. El clasificador de Bayes tiene un parámetro *alpha* (que se describirá posteriormente), el cuál tendrá un valor de 0,1 en esta prueba.

Para los *word embeddings* se prueban los conjuntos con una red con una capa convolucional de 128 filtros de tamaño 5 y una capa LSTM con 100 neuronas, incluyendo una capa previa para introducir los vectores preentrenados. Dado que las redes neuronales requieren un vector de entrada con un determinado tamaño, en este caso será de 100, es decir, se truncarán los vectores de palabras a una longitud de 100 palabras para poder utilizarlos en una red. Además, tras los embeddings se añade una capa de *dropout* con una tasa de 0,2 para reducir el sobreaprendizaje de la red y detrás de la capa convolucional se incluye una capa de reducción global. El entrenamiento de la red se hará en 3 ciclos de aprendizaje.

La experimentación con el modelo ULMFiT se realiza mediante la librería *fast.ai*, la cuál contiene la implementación del modelo, que consiste en una red con 3 capas LSTM que cuentan con un determinado dropout para normalizar el aprendizaje. Este modelo ha sido entrenado en un conjunto de datos de WikiText con embeddings de dimensión 400. A falta de concretar más, pues se detallará posteriormente, el modelo se encuentra configurado con los parámetros que mejores resultados han dado, por lo que para esta experimentación previa el modelo cuenta con 1150 neuronas en cada capa de LSTM y dropout en cada una de ellas. Por tanto, el parámetro a ajustar por ahora es la tasa de aprendizaje, la cuál es de 0.01 para ajustar el modelo preentrenado a nuestro vocabulario y 0.003 para la clasificación, realizando 3 ciclos de aprendizaje para evitar el sobreaprendizaje.

En primer lugar, se va a observar la diferencia entre el conjunto sin stemmatizar y el stemmatizado. Ambos conjuntos se prueban de la misma forma con los 3 modelos propuestos sin alterar ningún parámetro.

TABLA 4.1. EXPERIMENTACIÓN PREVIA CON
CONJUNTO STEMATIZADO

Conjunto de datos	Clasificador					
	Naïve Bayes		CNN+LSTM		ULMFiT	
	Binario	Multi.	Binario	Multi.	Binario	Multi.
Conjunto limpio	80, 43	51, 33	85, 89	58, 94	85, 25	55, 61
Conjunto stemmatizado	80, 01	50, 94	83, 39	55, 68	84, 30	54, 55

Los resultados están expresados como el porcentaje de la exactitud.

Visualizando los resultados de la tabla 4.1 y habiendo probado los clasificadores de la misma forma para los distintos conjuntos, se aprecia que los resultados con el conjunto stemmatizado son peores que los del conjunto sin stemmatizar, tanto para el caso de la clasificación binaria como para la multiclase. La mayor diferencia es del 3,26 % y la menor es del 0,39 %, no habiendo ningún caso en el que el stemmatizado sea mejor. Llegando a esta conclusión, se empleará a lo largo de la experimentación el conjunto limpio que no ha sido stemmatizado. La razón que puede estar detrás de estos resultados ya se ha mencionado: al stemmatizar el texto, es posible que se junten palabras que aporten distintos significados a la reseña y para los vectores preentrenados se puede dar que los vectores no hayan sido stemmatizados y no se ajusten bien a este conjunto, siendo así el peor resultado.

Por tanto, se pasa a comprobar el resultado de utilizar un conjunto de datos con el doble de reseñas.

TABLA 4.2. EXPERIMENTACIÓN PREVIA CON EL DOBLE
DE DATOS

Conjunto de datos	Clasificador					
	Naïve Bayes		CNN+LSTM		ULMFiT	
	Binario	Multi.	Binario	Multi.	Binario	Multi.
125000 reseñas	80, 43	51, 33	85, 89	58, 94	85, 25	55, 61
250000 reseñas	81, 06	52, 20	86, 05	59, 39	85, 56	56, 21

Los resultados están expresados como el porcentaje de la exactitud.

Como es lógico, utilizar el doble de reseñas da lugar a mejores resultados, consiguiendo hasta un 0,87 % de mejora, aunque este resultado podría ser mayor si se incrementasen los ciclos de aprendizaje. El problema es que utilizar el doble de reseñas implica aumentar considerablemente el tiempo de entrenamiento, incluso doblándolo, lo cuál no es viable en el contexto de este trabajo, teniendo en cuenta también que los modelos de redes neuronales profundas conllevan mucho tiempo de cómputo. Por este motivo, se ha decidido que la mejora de doblar el número de reseñas no es suficiente, por lo que se realizará la experimentación con el conjunto de 125000 reseñas.

4.2. Experimentación con el modelo bag-of-words

En esta sección se describe la experimentación que se realiza con los distintos algoritmos de aprendizaje automático descritos en el capítulo de “Estado de la cuestión”, explicando el proceso que se realiza para encontrar los parámetros que mejores resultados producen sobre cada conjunto de datos. Concretamente, los algoritmos con los cuáles se va a experimentar son *Naïve Bayes* (Bayes ingenuo), las Máquinas de soporte vectorial, Random Forest y la Regresión Logística.

Usualmente, para escoger los parámetros que mejor funcionan para un problema se realiza una separación del conjunto de datos en un conjunto de entrenamiento, que corresponde al 70-80 % del conjunto total, para entrenar el modelo, y un conjunto de validación o test, un 20-30 %, que sirve para validar la capacidad de generalización del modelo, es decir, lo bien que funcionan los parámetros escogidos en un conjunto diferente al entrenado. En este caso, en lugar de hacer este tipo de validación de los parámetros, se realiza una **validación cruzada**.

La razón de utilizar validación cruzada es que la división en entrenamiento y test aún puede no reflejar la generalización del problema si los conjuntos están muy relacionados. Para evitar esto, la validación cruzada consiste en dividir el conjunto en K subconjuntos (en nuestro caso serán 5), de forma que uno de los conjuntos es de validación y el resto de entrenamiento. Durante el proceso, se realiza K veces el entrenamiento, cambiando el conjunto de validación por uno que antes era de

entrenamiento, empleando así distintos conjuntos para validar y al final se realiza la media de las K iteraciones para determinar el resultado del modelo. La validación cruzada, debido a sus iteraciones, es más costosa en tiempo, pero se ha escogido porque este tipo de algoritmos no son muy lentos de entrenar y entrenar de esta manera produce mejores resultados.

Existen algunos factores comunes a todos los algoritmos. Por un lado, se probará a vectorizar las palabras convirtiéndolas tanto en vectores de ocurrencias como en vectores de frecuencias inversas, probando de qué forma se obtienen mejores resultados. Además, existe la posibilidad de ajustar la frecuencia de los términos de forma sublineal, cambiando la frecuencia tf por $1 + \log(tf)$ (Pedregosa et al., 2011). Por otra parte, esta vectorización se realizará de las palabras por sí solas (unigramas) y de las palabras teniendo en cuenta sus palabras adyacentes para que el algoritmo considere de alguna manera el contexto de la palabra (bigramas).

El planteamiento de los parámetros a experimentar se realizará mediante búsqueda organizada por cuadrícula, es decir, se plantea una cuadrícula de parámetros que se prueban con los distintos algoritmos, seguido de una búsqueda manual si se encuentran tendencias destacables.

Clasificador Bayesiano ingenuo

La experimentación con Bayes ingenuo se realizará a través de su variante multinomial, cuya implementación ofrece la librería Scikit-learn con el módulo *MultinomialNB*. Esta distribución se emplea porque, como se indicó en el apartado 2.2.1, es la variante que mejor funciona en problemas de clasificación de texto.

Este algoritmo es bastante simple, por lo que la importancia de su resultado reside, principalmente, en la vectorización de las palabras. No obstante, cuenta con un parámetro de suavizado:

- **alpha:** es un parámetro de suavizado cuyo valor puede encontrarse entre 0 y 1. Dicho suavizado se llama suavizado aditivo o suavizado de Laplace y, dado que este algoritmo está basado en la probabilidad, el suavizado hace que

el algoritmo asuma que todos los ngramas se han visto al menos una vez en todas las clases, evitando que la probabilidad de los ngramas que no se han visto sea nula para no perder información.

De esta forma, en la siguiente tabla se pueden apreciar las distintas posibilidades que se van a experimentar a priori.

TABLA 4.3. PARÁMETROS DE EXPERIMENTACIÓN CON
NAÏVE BAYES

Parámetro	Valores a experimentar
Vectorización	Ocurrencias Frecuencias inversas
Ngramas	Unigramas Unigramas y bigramas
Frecuencia sublineal	Sí No
Alpha	1 0,5 0,1 0,01 0,001

En base a estos parámetros se realizará una búsqueda exhaustiva, dado que el tiempo de cómputo de este algoritmo no es mucho. Si se observa una posible tendencia a partir de alguno de los parámetros, se profundizará en la misma para lograr el mejor posible resultado. Como punto de partida, se empleará el algoritmo con vectorización por ocurrencias, unigramas, sin frecuencia sublineal y con $\alpha = 0,1$.

Máquinas de soporte vectorial

En esta experimentación se va a usar el módulo *SVM* de la librería Scikit-learn, que contiene diversas implementaciones de métodos de máquinas de soporte vectorial y, en concreto, la clase *SVC*, un clasificador con máquinas de soporte vectorial. La función kernel que se va a emplear, de las indicadas en el apartado 2.2.1, es la función lineal, pues la complejidad temporal del resto es demasiado alta.

Para el entrenamiento de este algoritmo se cuentan con el siguiente parámetro:

- **C**: este parámetro es un parámetro de ajuste similar a la tasa de aprendizaje en las redes neuronales, es decir, penaliza el error de entrenamiento, controlando la

influencia de los vectores de soporte. Un valor pequeño de C implica un valor margen de los vectores, con una función más simple pero menor precisión, mientras que un valor mayor hará que la función sea más compleja y se ajuste mejor a los datos.

Es necesario encontrar el valor ideal de este parámetro para nuestro problema. Teniendo en cuenta este y los parámetros comunes, se plantean las posibilidades de experimento representadas en la tabla 4.4.

Al igual que en el algoritmo anterior y en los próximos, los parámetros de esta tabla son orientativos y se modificarán en caso de descubrir una tendencia que ayude a encontrar los parámetros ideales. Para comenzar la experimentación, se tomará como inicio una vectorización por ocurrencias, unigramas, sin frecuencia sublineal y con $C = 0,1$.

TABLA 4.4. PARÁMETROS DE EXPERIMENTACIÓN CON SVM

Parámetro	Valores a experimentar
Vectorización	Ocurrencias Frecuencias inversas
Ngramas	Unigramas Unigramas y bigramas
Frecuencia sublineal	Sí No
C	0,01 0,1 1 10 50

Random Forest

Los árboles de decisión no tienen parámetros que ajustar como tal, sino parámetros para modelar el tamaño de los mismos y aumentar o reducir el rendimiento del modelo. Al ser una técnica de *bagging*, Random Forest se basa en la composición de árboles de decisión, por lo que lo único que se experimentará es la variación del número de árboles que se utilizan para la clasificación. Además, este modelo es computacionalmente muy exigente, pues se tienen que entrenar múltiples clasificadores, así que no se podrá incrementar el número de árboles excesivamente.

Así las cosas, los parámetros de experimentación planeados para este modelo serían

los siguientes:

TABLA 4.5. PARÁMETROS DE EXPERIMENTACIÓN CON
RANDOM FOREST

Parámetro	Valores a experimentar
Vectorización	Ocurrencias Frecuencias inversas
Ngramas	Unigramas Unigramas y bigramas
Frecuencia sublineal	Sí No
Número de árboles	50 100 200

Regresión logística

La regresión logística es un algoritmo muy similar a Naïve Bayes, por lo que su funcionamiento no es muy complejo. Para su entrenamiento se utiliza un algoritmo de optimización, existiendo distintas posibilidades de las cuáles se ha escogido la función *sag* (*Stochastic Averaged Gradient*), pues es la que mejor funciona para conjuntos grandes y para problemas con clasificación multiclase (Pedregosa et al., 2011). La implementación de este modelo que se va a usar es la de la clase *LogisticRegression* de Scikit-learn y este algoritmo cuenta con un único parámetro que ajustar:

- **C**: este parámetro es muy similar al parámetro C de las máquinas de soporte vectorial, es un parámetro de regularización de la función que regulariza más la curva con valores pequeños.

Como en los anteriores algoritmos, la planificación resultante contiene una serie de parámetros entre los que se va tratar de hallar el modelo ideal:

TABLA 4.6. PARÁMETROS DE EXPERIMENTACIÓN CON
REGRESIÓN LOGÍSTICA

Parámetro	Valores a experimentar
Vectorización	Ocurrencias Frecuencias inversas
Ngramas	Unigramas Unigramas y bigramas
Frecuencia sublineal	Sí No
C	0,001 0,1 1 10 100 500

4.3. Experimentación con word embeddings

A continuación, se detalla la experimentación que se ha realizado para probar con modelos que empleen word embeddings, los cuáles se desarrollan mediante aprendizaje profundo.

Los modelos de aprendizaje profundo, al contar con una mayor cantidad de capas y neuronas que las redes de neuronas tradicionales, son más costosos de entrenar, sobre todo teniendo en cuenta que el conjunto de datos de este problema es bastante grande (aún habiendo sido reducido). Por este motivo, se probará únicamente con dos distintas arquitecturas basadas en el estado del arte que hayan obtenido buenos resultados.

Al entrenar un modelo neuronal, se tienen dos objetivos principales: obtener la mayor exactitud posible durante el entrenamiento y en la validación (al igual que en el resto de algoritmos), y reducir al mínimo posible el error de la red. La exactitud no es más que el porcentaje de acierto de la red al predecir los resultados, mientras que el error es una medida de los errores cometidos al clasificar cada ejemplo. La experimentación no trata únicamente de mejorar la precisión, pues puede ocurrir que al mejorar ésta el error aumente en el conjunto de validación, lo cuál significa que se ha producido sobreaprendizaje, es decir, que los pesos de las neuronas se hayan ajustado demasiado a los ejemplos de entrenamiento y no se adapten bien a unos nuevos datos. Por tanto, para lograr mejorar la precisión y reducir el error al mínimo, se debe experimentar con ciertos parámetros que aumenten la primera

sin producir sobreaprendizaje y, al igual que en el resto de modelos de aprendizaje automático, no existe una regla de oro para encontrarlos, únicamente la realización de pruebas.

Para la validación de los parámetros se realizará un proceso de validación cruzada como en la experimentación del modelo anterior: se divide el conjunto de datos en 5 partes y se realizan 5 iteraciones distintas, tomando en cada iteración una de estas partes como conjunto para validar y el resto como conjunto de entrenamiento; al final del proceso, se toma la media del resultado de las 5 iteraciones.

Así, los parámetros relativos a las redes neuronales profundas son los siguientes:

- **Arquitectura:** en este problema se van a plantear como solución 2 posibles arquitecturas, pues probar todos los parámetros con más arquitecturas sería demasiado costoso. No obstante, las distintas configuraciones posibles pueden producir resultados muy distintos.
- **Tasa de aprendizaje:** es quizá uno de los parámetros más importantes de las redes de neuronas. Esta tasa cuantifica la modificación de los pesos en el algoritmo de retropropagación, de forma que una tasa mayor modifica mayormente los pesos y una tasa menor, al contrario. Si la tasa es demasiado grande, hará que la red aprenda más rápido modificando los pesos pero sea más difícil de optimizar (encontrar el mínimo global). Si la tasa es demasiado pequeña, los pesos no se modificarán mucho, teniendo que entrenar la red durante más tiempo y con la posibilidad de que se quede en un mínimo local. Por ello, se tratará de dar con la tasa óptima probando con distintos valores. Asimismo, para evitar el sobreaprendizaje, se aplicará un **decremento de la tasa de aprendizaje** para que ésta se reduzca a medida que se realiza el entrenamiento.
- **Función de optimización:** la función de optimización, de la cual la tasa de aprendizaje es un parámetro, es la función encargada de encontrar el mínimo de la función de error. Existen distintas funciones, por lo que se probará tanto con la función Adam como con la función SGD (*Stochastic Gradient Descent*, dado que al optimizar la función de error de distintas formas pueden ofrecer

diversos resultados.

- **Ciclos de aprendizaje:** los ciclos de aprendizaje son el número de pasadas que realizará la red por el conjunto de datos. Este parámetro está ligado a la tasa de aprendizaje, pues cuanto más ciclos se realicen, mayor efecto tendrá la tasa. Por ello, lo ideal es hacer el máximo número de ciclos posibles para que entrene la red, siempre que sea un tiempo de cómputo aceptable, mientras no exista sobreaprendizaje, en cuyo caso se debería limitar el número de ciclos o ajustar el resto de parámetros. En base a esto, el número de ciclos vendrá determinado por la condición de parada, que será el error de la red, es decir, se realizará *early stopping*, parando el número de ciclos del experimento una vez comience a aumentar el error.
- **Número de neuronas:** aunque la arquitectura será fija, se prueba con distintas unidades de neuronas en las capas internas, pues al realizarse más procesamiento pueden aparecer mejores resultados.

Estos son los parámetros principales con los que experimentar, aunque existen otros que también tienen influencia en el aprendizaje de las redes. Además, existe una limitación al ser un problema de clasificación de texto, y es que las redes neuronales cuentan con una entrada fija y cada reseña tiene una longitud diferente. Por ello, se deberá limitar el tamaño de la entrada, lo cual se hará en base a las longitudes medias vistas en la tabla 3.2. Tener en cuenta muchas palabras es complicado a nivel computacional, por lo que se reducirá el tamaño y a su vez se irá probando con **distintos tamaños de entrada**.

A la hora de buscar estos parámetros óptimos para nuestro problema existen varias opciones: búsqueda aleatoria, que consiste en probar un determinado número de parámetros aleatorios con el fin de encontrar los que mejores resultados den; búsqueda organizada por cuadrícula, que trata de establecer una serie de parámetros en cuadrícula y probarlos todos para encontrar el que mejor resultados de; y búsqueda manual, que se basa en seleccionar una serie de parámetros e ir probando en base a suposiciones. En esta experimentación, en un principio, se partirá de los parámetros que se han comentado en las publicaciones en las que se basan las arquitecturas,

pues son las que mejor les han funcionado a los creadores, para después detectar de qué forma se comportan los parámetros en nuestro problema y modificarlos con un planteamiento manual.

El proceso de experimentación sería el siguiente proceso iterativo:

1. Se plantea un experimento con unos parámetros, centrando el foco en un parámetro determinado.
2. Se entrena la red con esos parámetros.
3. Se observa el resultado de la red y, en base al mismo, se sigue la tendencia positiva, si la hay, o se cambia si es negativa.
4. Se vuelve al paso 1, cambiando el parámetro fijado.

La experimentación se prolongará el máximo tiempo posible para encontrar el mejor resultado y, en el paso 3, no solo se observará el resultado de las predicciones sino también el error de la función, pues se deberá corregir si existe sobreaprendizaje.

Como se ha podido ver en el estado de la cuestión, las publicaciones que mejores modelos han estudiado destacan una tendencia dominante que refleja que las redes que mejor funcionan en este tipo de problemas son las redes convolucionales y las redes con LSTM. Por esta razón, se trata de replicar los resultados de alguna de las investigaciones vistas, utilizando arquitecturas basadas en ellas para experimentar con *word embeddings*. Teniendo en mente el proceso de experimentación, a continuación se describen las arquitecturas a emplear.

Red neuronal convolucional profunda en forma piramidal

Tal y como se veía en la tabla 2.2, el mejor resultado que se ha encontrado en la literatura corresponde a Johnson y Zhang (2017), con el modelo que plantean de red convolucional. Se llama piramidal debido a que consiste en la composición de filtros convolucionales con capas de reducción, de forma que las características de entrada se van reduciendo conforme se avanza en la red, formando una pirámide.

La arquitectura (que se puede ver en la figura 4.1) consiste, básicamente, en bloques

de capas convolucionales con 3 filtros de 250 de tamaño seguidos de capas de reducción con tamaño 3 y paso 2 para reducir las representaciones del texto. A priori, es una red simple (aunque muy profunda), pero es la que mejores resultados ha proporcionado al análisis de sentimiento.

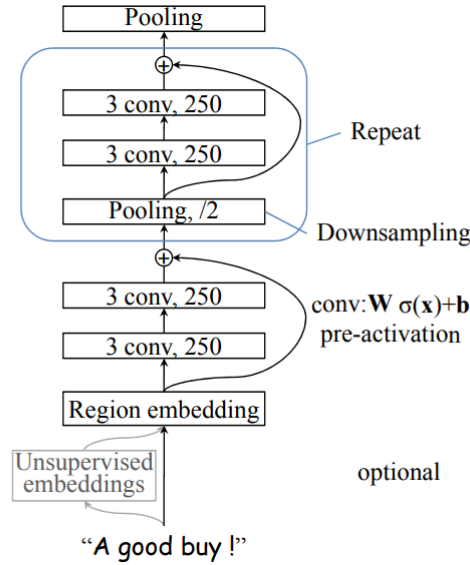


Fig. 4.1. Red neuronal convolucional profunda en forma piramidal (Johnson & Zhang, 2017)

Aunque la idea de la red es la repetición de los bloques convolucionales indefinidamente, probando hasta que se encuentre el número de bloques que mejor resultado provea, para acotar la experimentación se va a realizar una implementación de 7 bloques, debido a que éste es el número de bloques que los investigadores han encontrado óptimo (Johnson & Zhang, 2017). La función optimizadora es Adam, que comienza con una tasa de aprendizaje de 0,001 y se aplica una regularización l2 con una tasa de 0,00001.

Las capas convolucionales se incluyen mediante la capa *Conv1D* de Keras, con *filters* = 3 y *kernel_size* = 250, seguidas de las capas de reducción *MaxPooling1D* con *pool_size* = 3 y *strides* = 2. Igualmente, se añaden los parámetros comentados en el párrafo anterior mediante el optimizador *adam* de Keras y el regularizador *l2*.

Partiendo de este punto se tratará de encontrar un experimento que se acerque a

los resultados del modelo.

LSTM bidireccional con reducción en dos dimensiones

Aunque han aparecido muchos modelos convolucionales como el anterior que han tratado de conseguir el éxito que han logrado este tipo de redes en la clasificación de imágenes, la tendencia predominante dentro de la clasificación de texto ha estado en las redes recurrentes y, más concretamente, en las redes LSTM.

Los propios Johnson y Zhang, que desarrollaron el modelo anterior, plantearon otro modelo anteriormente (Johnson & Zhang, 2016) que emplea de forma muy simple y con muy buenos resultados una capa de LSTM bidireccional seguida de una capa de reducción. Por otro lado, se tiene el modelo de Zhou et al. (2016) (el que se va a probar), que utiliza una capa inicial de LSTM bidireccional que toma los *embeddings* y aprende información sobre las secuencias de textos a lo largo del tiempo. De esta capa se pasa a una capa convolucional de 2 dimensiones que termina en otra capa de reducción en 2 dimensiones.

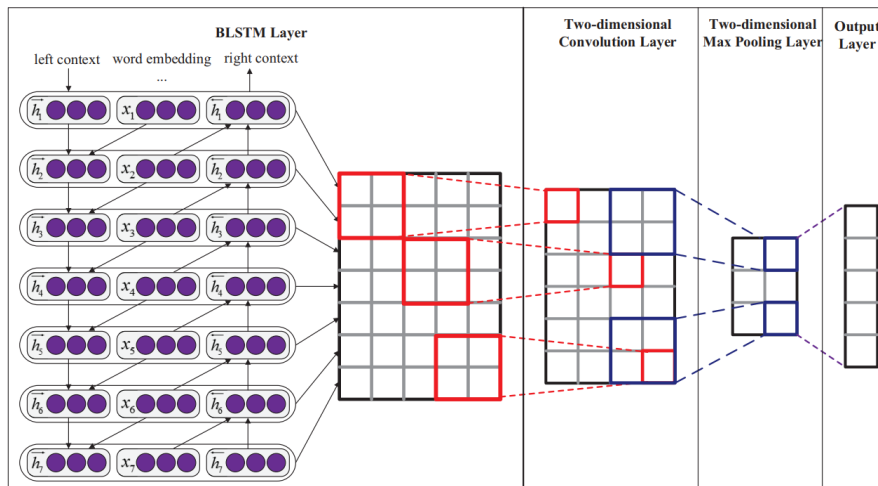


Fig. 4.2. LSTM bidireccional con reducción en dos dimensiones (Zhou et al., 2016)

La capa bidireccional cuenta con 5 unidades de neuronas ocultas, lo cuál se implementa mediante la capa *Bidirectional(LSTM)* de Keras con *units* = 5. Las capas de convolución y reducción son de 2 dimensiones con reducción (2, 2), implementándose,

respectivamente, con *Conv2D* y *MaxPooling2D*.

De acuerdo a la publicación, los parámetros óptimos para el investigador son: 300 neuronas en la capa LSTM, con 100 filtros convolucionales de tamaño 3 y reducciones a la mitad (2,2). Además, utiliza el optimizador *AdaDelta* con una tasa de aprendizaje de 1,0 y añade dropout⁹ de 0,5 a los *embeddings*, 0,2 a la capa bidireccional y 0,4 a la capa convolucional. Utiliza también un regularizador *l2* con una tasa de 0,00001.

Se considera muy favorable que en este modelo los investigadores proporcionen los parámetros óptimos para ayudar a replicar su modelo.

4.4. Experimentación con ULMFiT

Por último se realiza la experimentación con el modelo ULMFiT.

Aunque sea un modelo que consiste en “universalizar” los *word embeddings*, al fin y al cabo se trata de un modelo de red neuronal profundo con una determinada arquitectura que utiliza vectores preentrenados. Por este motivo, toda la experimentación descrita en el apartado anterior se aplica también para este modelo.

La arquitectura que emplean en su modelo es una arquitectura llamada AWD-LSTM, planteada por Merity, Keskar y Socher (2017). Los buenos resultados de este modelo están producidos por la utilización de esta arquitectura junto al entrenamiento de los vectores de *embedding* en un corpus de WikiText de más de 100 millones de palabras. La estructura es una conexión de 3 capas simples de LSTM con dropout y la clave está en la forma de entrenar el modelo: primero se toman los *embeddings* entrenados y se realiza un “ajuste fino” (*fine tuning*) sobre los datos de este problema, y después se pasa a entrenar el clasificador.

Para este entrenamiento se emplean dos técnicas novedosas: ajuste fino discriminatorio (*discriminative fine tuning*), que consiste en aplicar una tasa de aprendizaje

⁹el dropout es una técnica de regularización que consiste en dejar de tener en cuenta ciertas unidades de neuronas para controlar el aprendizaje (Hinton, Srivastava, Krizhevsky, Sutskever & Salakhutdinov, 2012)

distinta a cada capa, debido a que cada capa guarda una determinada información; y descongelación gradual del clasificador (*gradual unfreezing*), que consiste en congelar las capas (evitar que sus pesos se actualicen) y descongelarlas gradualmente desde la última capa, que es la que tiene información más “general” (Howard & Ruder, 2018).

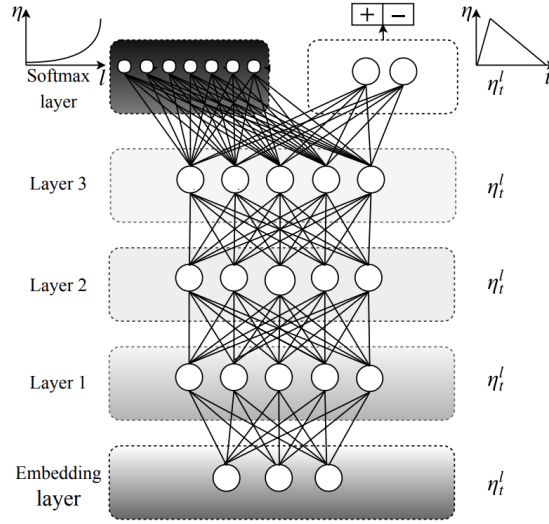


Fig. 4.3. Arquitectura AWD-LSTM (Merity et al., 2017)

La librería *fast.ai* está desarrollada por los creadores del modelo, por lo que incluye su implementación, poniendo al alcance también los métodos de *freeze()* y *unfreeze()* para realizar la descongelación gradual y permite entrenar las capas una a una para el ajuste discriminatorio, por lo que se experimentará con estas técnicas para lograr el mejor resultado.

Los parámetros óptimos de acuerdo a la publicación, y por los cuáles se comenzará a probar, son: 3 capas de LSTM con 1150 neuronas ocultas por capa para el ajuste fino del modelo, un dropout de 0,4 a la capa de *embedding* y de 0,3 a las capas recurrentes, y el optimizador Adam con tasa de aprendizaje 0,004, mientras que para la clasificación se cuentan con 50 neuronas en cada capa y una tasa de aprendizaje de 0,01 con el mismo optimizador.

5. EVALUACIÓN DE RESULTADOS

En este capítulo, el último relativo al estudio práctico de este trabajo, se indican los resultados obtenidos tras la realización de la experimentación detallada en el capítulo anterior y la elección de los modelos de acuerdo a dichos resultados.

5.1. Métricas de evaluación

De cara a escoger uno de los modelos resultantes, es necesaria la aplicación de métricas que midan el desempeño de cada modelo y nos permitan conocer mediante un valor lo bien/mal que han funcionado. Existen muchas métricas para evaluar el resultado de un entrenamiento, por lo que se realizará mediante las métricas más comunes (Pedregosa et al., 2011):

- **Accuracy** (exactitud): es la métrica más simple y a su vez la más empleada para evaluar modelos. La exactitud es el porcentaje de predicciones correctas sobre el número total de predicciones (ecuación 5.1). Cuando el número de ejemplos por categoría está desbalanceado, habría que corregir la exactitud para que tome en cuenta el número de ejemplos, pero en nuestro caso todas las categorías tienen el mismo número de reseñas.

$$accuracy = \frac{\#predicciones\ correctas}{\#total\ de\ predicciones} \quad (5.1)$$

- **Valor F1**: el valor F1 es el promedio de otras dos métricas similares a la exactitud: la precisión y la exhaustividad. La precisión es el porcentaje de reseñas de una estrella correctamente predichas sobre el total de predicciones de esa estrella, es decir, mide el porcentaje de veces en que las reseñas clasificadas con una estrella determinada correspondan de verdad a esa estrella. La exhaustividad es el porcentaje de predicciones correctas para un estrella sobre el total de reseñas de esa estrella, es decir, mide el porcentaje de veces que se identificaron correctamente las reseñas de esa estrella. El valor F1 no es más que una medida que combina ambas métricas, pues un buen modelo debe tener

buena precisión y también buena exhaustividad, viéndose la media de ambos con el valor F1:

$$f1 = \frac{2 * precision * exhaustividad}{precision + exhaustividad} \quad (5.2)$$

- **Matriz de confusión:** aunque no se utilizará para comparar modelos, se empleará la matriz de confusión para evaluar el desempeño de los mejores modelos. Esta matriz representa de que manera se ha clasificado cada categoría de la siguiente manera:

TABLA 5.1. MATRIZ DE CONFUSIÓN

		Predicción	
		Positivo	Negativo
Categoría correcta	Positivo	Verdadero positivo	Falso positivo
	Negativo	Falso negativo	Verdadero negativo

Por lo que lo ideal es que la diagonal central tenga el mayor número de valores, pues serían clasificaciones correctas, y además nos permite detectar si hay alguna categoría confundiéndose con otra.

- **ROC-AUC** (área bajo la curva de característica operativa del recepto): la curva ROC¹⁰ es un gráfico de una función que representa la tasa de verdaderos positivos frente a la tasa de falsos positivos. Por tanto, AUC¹¹, área bajo la curva, mide el área que se encuentra debajo de la curva ROC (en un plano bidimensional), es decir, una integral de (0,0) a (1,1). Por tanto, cuantas más predicciones sean correctas, más se acercará el AUC a 1 y viceversa.

Hasta ahora, todas las métricas que se han planteado son métricas que se deben **maximizar**, es decir, cuanto mayor sean mejor es el modelo evaluado. No obstante, se va a medir también el error o coste de los modelos, la cuál es una métrica a **minimizar**.

- **Cross-entropy loss:** el coste/error logarítmico o de entropía cruzada mide el desempeño del modelo respecto a una predicción entre 0 y 1. Esto es, evalúa

¹⁰Receiver Operating Characteristic

¹¹Area Under the Curve

el error que se comete al realizar predicciones, de forma que el error aumenta conforme las predicciones se alejan de la categoría correspondiente. Así, el objetivo de un modelo es reducir el error al mínimo, lo que significa que las predicciones se acercan a la categoría debida, por lo que un modelo perfecto tendría un coste 0. La ecuación para su cálculo sería la siguiente:

$$-(y \log(p) + (1 - y) \log(1 - p)) \quad (5.3)$$

donde y es el valor de la categoría y p es la categoría predicha. Para el caso de la clasificación multiclase, se calcula el coste para cada categoría y se suma, formando el total del coste del modelo.

Estas métricas se utilizan para evaluar los resultados de cada uno de los experimentos, los cuáles se desglosan a continuación.

5.2. Evaluación del modelo bag-of-words

En esta sección se procede a evaluar los resultados de la experimentación con los algoritmos que utilizan el modelo *bag-of-words* y cuyo proceso de experimentación se ha descrito en el capítulo anterior.

Los resultados están ordenados en base a la exactitud de los experimentos, siendo la primera fila de la tabla la que representa el resultado con la mejor exactitud.

5.2.1. Bayes ingenuo

Para la experimentación con un clasificador bayesiano ingenuo se emplean los parámetros especificados en la tabla 4.3. Los resultados para el conjunto de datos binario son los siguientes:

TABLA 5.2. RESULTADOS DE BAYES INGENUO EN EL
CONJUNTO BINARIO

Parámetros				Resultados			
Alpha	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
0,5	Uni. y bi.	TF-IDF	Sí	83,99	83,61	0,9194	0,1675
0,5	Uni. y bi.	TF-IDF	No	83,86	83,47	0,9184	0,1728
0,1	Uni. y bi.	TF	Sí	83,83	83,79	0,9185	0,0957
0,1	Uni. y bi.	TF-IDF	Sí	83,82	83,80	0,9175	0,0341
1	Uni. y bi.	TF-IDF	Sí	83,79	83,23	0,9186	0,2500
0,1	Uni. y bi.	TF	No	83,77	83,74	0,9184	0,1048
0,1	Uni. y bi.	TF-IDF	No	83,74	83,70	0,9170	0,0367
1	Uni. y bi.	TF-IDF	No	83,65	83,07	0,9173	0,2546
0,5	Uni. y bi.	TF	Sí	83,56	83,01	0,9167	0,2416
0,5	Uni. y bi.	TF	No	83,48	82,97	0,9163	0,2499
0,01	Uni. y bi.	TF	Sí	83,43	83,51	0,9137	0,0151
0,01	Uni. y bi.	TF	No	83,41	83,49	0,9143	0,0168
1	Uni. y bi.	TF	Sí	83,25	82,45	0,9149	0,3014
1	Uni. y bi.	TF	No	83,20	82,48	0,9143	0,3075
0,01	Uni. y bi.	TF-IDF	Sí	82,54	82,50	0,9055	0,0039
0,01	Uni. y bi.	TF-IDF	No	82,52	82,47	0,9057	0,0040
0,001	Uni. y bi.	TF	No	82,48	82,34	0,9049	0,0055
0,001	Uni. y bi.	TF	Sí	82,42	82,28	0,9039	0,0052
0,5	Unigramas	TF	Sí	81,67	81,63	0,8987	0,3888
0,5	Unigramas	TF	No	81,62	81,63	0,8989	0,3913
1	Unigramas	TF	Sí	81,62	81,32	0,8986	0,4020
0,1	Unigramas	TF	Sí	81,61	81,87	0,8981	0,3513
1	Unigramas	TF	No	81,61	81,40	0,8988	0,4037
0,1	Unigramas	TF	No	81,55	81,82	0,8984	0,3561
1	Unigramas	TF-IDF	Sí	81,30	80,96	0,8949	0,3771
1	Unigramas	TF-IDF	No	81,28	80,99	0,8947	0,3801
0,01	Unigramas	TF	Sí	81,26	81,56	0,8946	0,2918
0,01	Unigramas	TF	No	81,25	81,57	0,8952	0,2988
0,5	Unigramas	TF-IDF	Sí	81,23	81,14	0,8937	0,3517
0,5	Unigramas	TF-IDF	No	81,20	81,14	0,8935	0,3557
0,001	Uni. y bi.	TF-IDF	No	80,89	80,53	0,8908	0,0018
0,001	Uni. y bi.	TF-IDF	Sí	80,88	80,51	0,8905	0,0018
0,1	Unigramas	TF-IDF	Sí	80,74	80,90	0,8886	0,2867

0,001	Unigramas	TF	No	80,73	81,01	0,8882	0,2577
0,001	Unigramas	TF	Sí	80,71	80,98	0,8873	0,2507
0,1	Unigramas	TF-IDF	No	80,68	80,86	0,8886	0,2929
0,01	Unigramas	TF-IDF	No	79,59	79,75	0,8743	0,2267
0,01	Unigramas	TF-IDF	Sí	79,59	79,72	0,8737	0,2210
0,001	Unigramas	TF-IDF	No	78,37	78,45	0,8549	0,1963
0,001	Unigramas	TF-IDF	Sí	78,31	78,37	0,8536	0,1922

A simple vista se puede observar que, aunque se ha probado con unigramas, en casi todos los casos los mejores resultados han sido con unigramas y bigramas. El mejor experimento alcanza un 83,99% de exactitud, lo cuál indica que se ha entrenado bien el modelo y que funciona bien para los conjuntos de validación. Igualmente, los valores F1 correspondientes son muy similares a los de la exactitud, lo que significa que la precisión y la exhaustividad del modelo están equilibradas. Los valores de *alpha* que mejor exactitud dan son de 0,1 y 0,5, siendo el error del modelo bastante bajo (0,1675), y aumentar el valor del parámetro aumenta demasiado el error de la red, mientras que reducirlo implica un error muy bajo pero a su vez una baja exactitud. De los 8 mejores experimentos, 6 de ellos emplean una vectorización por frecuencias inversas, por lo que se infiere que la combinación de unigramas y bigramas, vectorización en TF-IDF y valores de *alpha* 0,5 y 0,1 producen los mejores resultados para el problema. Sobre la modificación sublineal de la frecuencia no se puede extraer ninguna conclusión porque ofrece resultados diversos.

Además, el valor del área bajo la curva es cercano al 0,90 en la mayoría de experimentos, lo que demuestra que el modelo discrimina muy bien las reseñas de una categoría y otra. Si observamos la matriz de confusión del mejor experimento (figura 5.1), podemos ver que casi no se confunde al clasificar las reseñas, únicamente unas pocas reseñas positivas (1) las predice como negativas (0).

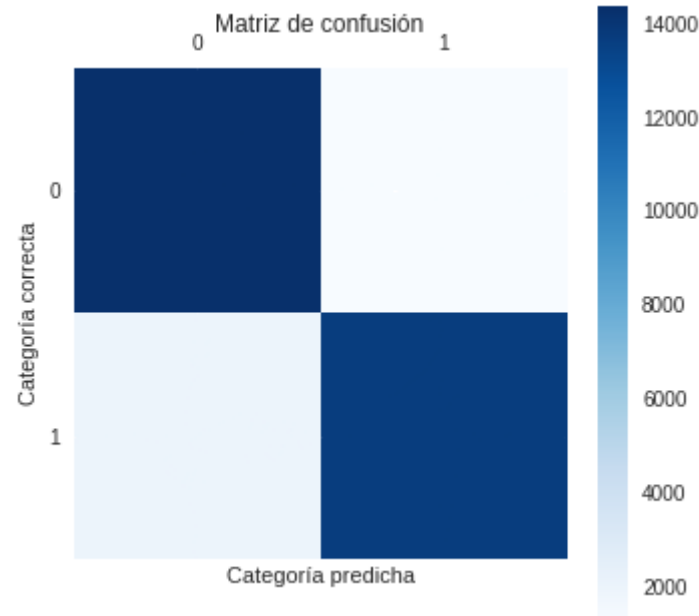


Fig. 5.1. Matriz de confusión de Naïve Bayes binario

En la tabla 5.3 se observan los resultados de este mismo clasificador para el conjunto multiclase.

TABLA 5.3. RESULTADOS DE BAYES INGENUO EN EL CONJUNTO MULTICLASE

Parámetros				Resultados			
Alpha	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
0,1	Uni. y bi.	TF	Sí	55,30	55,57	0,8053	1,0767
0,1	Uni. y bi.	TF	No	55,26	55,53	0,8063	1,0764
0,5	Uni. y bi.	TF-IDF	Sí	54,97	55,18	0,8225	1,0997
1	Uni. y bi.	TF-IDF	Sí	54,94	55,11	0,8241	1,1322
0,5	Uni. y bi.	TF-IDF	No	54,74	54,94	0,8005	1,1051
1	Uni. y bi.	TF-IDF	No	54,62	54,77	0,7966	1,1369
0,1	Uni. y bi.	TF-IDF	Sí	54,49	54,74	0,7845	1,0853
0,1	Uni. y bi.	TF-IDF	No	54,41	54,66	0,7747	1,0884
0,5	Uni. y bi.	TF	Sí	54,40	54,59	0,7535	1,1067
0,5	Uni. y bi.	TF	No	54,36	54,57	0,7725	1,1087
0,01	Uni. y bi.	TF	Sí	54,31	54,52	0,7859	1,1373
0,01	Uni. y bi.	TF	No	54,28	54,49	0,7561	1,1252
0,1	Unigramas	TF	Sí	53,96	54,02	0,7977	1,1363
0,1	Unigramas	TF	No	53,91	53,99	0,8149	1,1381

1	Uni. y bi.	TF	Sí	53,90	54,03	0,7996	1,1273
1	Uni. y bi.	TF	No	53,82	53,99	0,8155	1,1298
0,5	Unigramas	TF	No	53,24	53,43	0,8036	1,1546
0,5	Unigramas	TF	Sí	53,21	53,38	0,8037	1,1535
0,01	Unigramas	TF	No	53,06	53,09	0,8165	1,1454
0,01	Unigramas	TF	Sí	53,06	53,08	0,8166	1,1458
1	Unigramas	TF	No	52,52	52,71	0,7854	1,1692
1	Unigramas	TF	Sí	52,40	52,54	0,7846	1,1695
0,5	Unigramas	TF-IDF	Sí	52,24	52,37	0,7729	1,1626
0,5	Unigramas	TF-IDF	No	52,17	52,33	0,7619	1,1645
0,01	Uni. y bi.	TF-IDF	Sí	52,05	52,30	0,7630	1,271
0,01	Uni. y bi.	TF-IDF	No	52,03	52,27	0,7738	1,2629
0,001	Uni. y bi.	TF	No	51,98	52,29	0,7603	1,3097
0,001	Uni. y bi.	TF	Sí	51,88	52,20	0,7633	1,3418
1	Unigramas	TF-IDF	Sí	51,82	51,92	0,7223	1,1753
1	Unigramas	TF-IDF	No	51,75	51,89	0,7241	1,1768
0,001	Unigramas	TF	No	51,75	51,82	0,7166	1,1745
0,1	Unigramas	TF-IDF	Sí	51,72	51,76	0,7210	1,161
0,001	Unigramas	TF	Sí	51,71	51,77	0,7086	1,1789
0,1	Unigramas	TF-IDF	No	51,71	51,76	0,7096	1,1618
0,01	Unigramas	TF-IDF	No	49,71	49,74	0,6891	1,2193
0,01	Unigramas	TF-IDF	Sí	49,71	49,74	0,6916	1,2435
0,001	Uni. y bi.	TF-IDF	No	49,30	49,67	0,6849	1,6551
0,001	Uni. y bi.	TF-IDF	Sí	49,27	49,64	0,6857	1,6788
0,001	Unigramas	TF-IDF	Sí	48,14	48,23	0,6667	1,3476
0,001	Unigramas	TF-IDF	No	48,13	48,22	0,6698	1,3335

Los resultados que se obtienen son, como se esperaba, peores que al tratar de realizar la clasificación binaria, pues el modelo debe tratar de diferenciar las 5 estrellas. De nuevo, el uso de unigramas y bigramas es la tendencia más positiva, y los mejores valores de *alpha* son los mismos, 0,1 y 0,5. También se observa mucha variación en la vectorización en TF o en TF-IDF, por lo que este parámetro es más dependiente del resto de parámetros.

Se alcanza una exactitud de 55,30%, lo cuál es mejorable pero en el contexto de la

clasificación multiclase no es muy negativo. A su vez, se observa que el área bajo la curva no es tan buena (entre 0,66 y 0,81) y el error ha aumentado considerablemente respecto a la experimentación binaria.

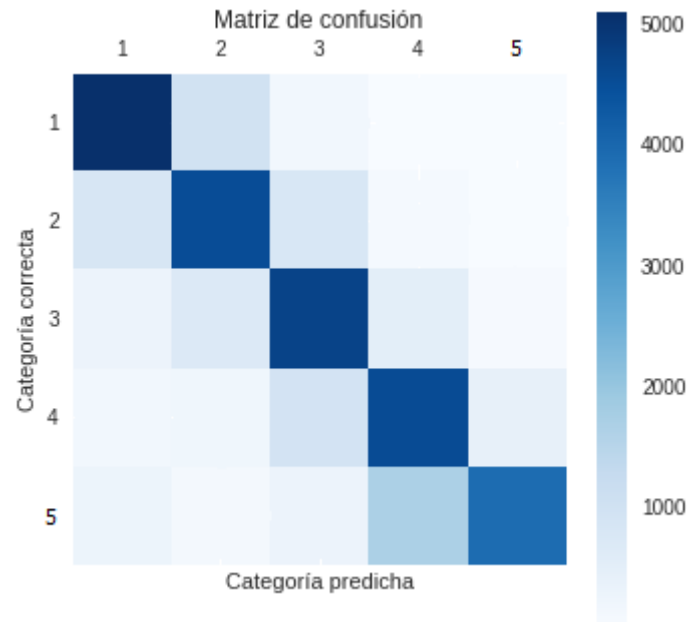


Fig. 5.2. Matriz de confusión de Naïve Bayes multiclase

Observando la matriz de confusión del mejor experimento, vemos que aunque la exactitud no es muy alta, en la mayoría de los casos se clasifican bien las estrellas y el modelo se equivoca prediciendo las estrellas colindantes, por lo que no hay un desvío severo.

5.2.2. SVM

La experimentación con máquinas de soporte vectorial en el conjunto binario ha dado lugar a los siguientes experimentos:

TABLA 5.4. RESULTADOS DE SVM EN EL CONJUNTO BINARIO

Parámetros				Resultados			
C	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
10	Uni. y bi.	TF-IDF	Sí	87,22	87,19	0,9452	0,3008

50	Uni. y bi.	TF-IDF	Sí	87,21	87,18	0,9451	0,3009
1	Uni. y bi.	TF-IDF	Sí	87,18	87,15	0,9451	0,3019
10	Uni. y bi.	TF-IDF	No	87,13	87,08	0,9444	0,3028
50	Uni. y bi.	TF-IDF	No	87,12	87,07	0,9443	0,3029
1	Uni. y bi.	TF-IDF	No	87,09	87,05	0,9443	0,3038
1	Uni. y bi.	TF	Sí	87,08	87,04	0,9444	0,3011
10	Uni. y bi.	TF	Sí	87,04	86,99	0,9437	0,3029
1	Uni. y bi.	TF	No	87,02	86,97	0,9437	0,3029
50	Uni. y bi.	TF	Sí	86,98	86,93	0,9433	0,3038
10	Uni. y bi.	TF	No	86,93	86,88	0,9429	0,3050
50	Uni. y bi.	TF	No	86,90	86,84	0,9425	0,3061
0,1	Unigramas	TF-IDF	Sí	86,22	86,19	0,9370	0,3205
0,1	Uni. y bi.	TF-IDF	Sí	86,14	86,12	0,9381	0,3219
0,1	Uni. y bi.	TF	Sí	86,09	86,06	0,9376	0,3191
1	Unigramas	TF	Sí	86,09	86,02	0,9371	0,3205
0,1	Uni. y bi.	TF-IDF	No	86,07	86,04	0,9371	0,3240
0,1	Unigramas	TF-IDF	No	86,05	86,01	0,9359	0,3232
1	Unigramas	TF	No	86,04	85,96	0,9364	0,3220
0,1	Uni. y bi.	TF	No	86,00	85,96	0,9367	0,3214
1	Unigramas	TF-IDF	Sí	85,87	85,81	0,9344	0,3275
0,1	Unigramas	TF	Sí	85,82	85,76	0,9353	0,3242
1	Unigramas	TF-IDF	No	85,77	85,70	0,9336	0,3294
0,1	Unigramas	TF	No	85,70	85,63	0,9342	0,3268
10	Unigramas	TF	Sí	85,04	84,96	0,9273	0,3470
10	Unigramas	TF	No	84,97	84,88	0,9269	0,3477
0,01	Unigramas	TF-IDF	Sí	84,50	84,52	0,9240	0,3514
0,01	Unigramas	TF-IDF	No	84,30	84,31	0,9220	0,3560
10	Unigramas	TF-IDF	Sí	84,01	83,97	0,9189	0,3669
10	Unigramas	TF-IDF	No	83,89	83,84	0,9183	0,3681
0,01	Unigramas	TF	Sí	83,79	83,76	0,9187	0,3619
50	Unigramas	TF	Sí	83,58	83,52	0,9151	0,3783
0,01	Unigramas	TF	No	83,56	83,51	0,9166	0,3664
50	Unigramas	TF	No	83,53	83,45	0,9146	0,3792
0,01	Uni. y bi.	TF	Sí	83,22	83,22	0,9140	0,3729
0,01	Uni. y bi.	TF-IDF	Sí	83,08	83,03	0,9132	0,3795
0,01	Uni. y bi.	TF	No	83,07	83,06	0,9123	0,3763
0,01	Uni. y bi.	TF-IDF	No	82,98	82,95	0,9115	0,3830

50	Unigramas	TF-IDF	Sí	82,42	82,39	0,9055	0,3990
50	Unigramas	TF-IDF	No	82,32	82,27	0,9046	0,4007

Se vuelve a ver que, principalmente, el uso de unigramas y bigramas supera con creces los resultados de usar unigramas, y que los cinco mejores experimentos vectorizan en frecuencias inversas, por lo que podría concluir que para las máquinas de soporte vectorial esta vectorización funciona mejor. Lo cuál tiene sentido, debido a que Bayes es un algoritmo que únicamente emplea probabilidades, mientras que SVM representa un espacio de vectores. El coste del modelo ha aumentado respecto al clasificador bayesiano, pero también se ve, curiosamente, que todas las métricas (exactitud, f1, roc-auc y error) mejoran de forma proporcional en todos los experimentos y que los valores de C que mejor funcionan son los más altos (50, 10 y 1). Esto último puede deberse a que un mayor valor de C hace que el hiperplano se ajuste mejor a los datos.

En vista de esto, aumentamos más el parámetro C para observar si se mantiene la tendencia en la exactitud:

TABLA 5.5. SEGUNDA EXPERIMENTACIÓN CON SVM EN EL CONJUNTO BINARIO

Parámetros				Resultados	
C	Ngramas	Vector	Sublin.	Acc. ↑	F1
100	Uni. y bi.	TF-IDF	Sí	87,19	87,15
100	Uni. y bi.	TF-IDF	No	87,06	87,01
100	Uni. y bi.	TF	Sí	86,78	86,74
100	Uni. y bi.	TF	No	86,63	86,59

Vemos que los resultados no han mejorado, pero se destaca la tendencia vista de que la frecuencia inversa funciona mejor para vectorizar las palabras en este problema.

En general, se obtienen buenos resultados para todas las métricas, con una mejor

exactitud de 87,22. Viendo la matriz de confusión queda claro que el modelo no tiene problemas en diferenciar ambas polaridades.

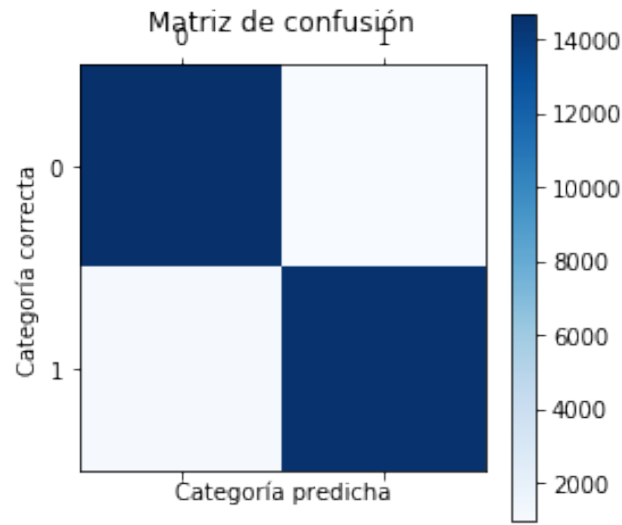


Fig. 5.3. Matriz de confusión de SVM binario

Los resultados con el conjunto multiclase son los siguientes:

TABLA 5.6. RESULTADOS DE SVM EN EL CONJUNTO MULTICLASE

Parámetros				Resultados			
C	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
1	Uni. y bi.	TF-IDF	Sí	59,65	59,25	0,8652	0,9658
1	Uni. y bi.	TF	Sí	59,45	59,05	0,8613	0,9740
1	Uni. y bi.	TF-IDF	No	59,39	58,93	0,8606	0,9701
1	Uni. y bi.	TF	No	59,29	58,79	0,8640	0,9777
0,1	Uni. y bi.	TF-IDF	Sí	59,19	58,60	0,8599	0,9752
10	Uni. y bi.	TF-IDF	Sí	59,08	58,83	0,8592	0,9796
0,1	Uni. y bi.	TF	Sí	59,03	58,46	0,8637	0,9799
0,1	Uni. y bi.	TF-IDF	No	58,98	58,30	0,8627	0,9798
50	Uni. y bi.	TF-IDF	Sí	58,96	58,69	0,8625	0,9820
0,1	Uni. y bi.	TF	No	58,88	58,10	0,8614	0,9846
10	Uni. y bi.	TF-IDF	No	58,84	58,52	0,8620	0,9842
50	Uni. y bi.	TF-IDF	No	58,73	58,39	0,8570	0,9868
10	Uni. y bi.	TF	Sí	58,53	58,15	0,8555	0,9984
10	Uni. y bi.	TF	No	58,34	57,83	0,8609	1,0030

50	Uni. y bi.	TF	Sí	58,33	57,85	0,8557	1,0040
0,1	Unigramas	TF-IDF	Sí	58,21	57,42	0,8540	1,0016
50	Uni. y bi.	TF	No	58,11	57,58	0,8561	1,0090
0,1	Unigramas	TF	Sí	58,10	57,23	0,8560	1,0025
0,1	Unigramas	TF-IDF	No	58,04	57,17	0,8549	1,0063
0,1	Unigramas	TF	No	57,93	57,04	0,8549	1,0068
1	Unigramas	TF	Sí	57,36	56,65	0,8498	1,0229
1	Unigramas	TF	No	57,28	56,44	0,8492	1,0248
0,01	Unigramas	TF-IDF	Sí	57,05	56,12	0,8419	1,0361
0,01	Unigramas	TF-IDF	No	56,75	55,77	0,8411	1,0443
1	Unigramas	TF-IDF	Sí	56,45	55,49	0,8407	1,0482
1	Unigramas	TF-IDF	No	56,33	55,44	0,8401	1,0509
0,01	Uni. y bi.	TF-IDF	Sí	56,22	55,48	0,8470	1,0522
0,01	Unigramas	TF	Sí	56,21	55,18	0,8432	1,0507
0,01	Unigramas	TF	No	55,96	54,91	0,8393	1,0580
0,01	Uni. y bi.	TF	Sí	55,92	55,07	0,8415	1,0567
0,01	Uni. y bi.	TF-IDF	No	55,88	55,10	0,8381	1,0611
0,01	Uni. y bi.	TF	No	55,72	54,84	0,8452	1,0633
10	Unigramas	TF	Sí	54,81	53,68	0,8279	1,0998
10	Unigramas	TF	No	54,69	53,43	0,8276	1,1005
10	Unigramas	TF-IDF	Sí	52,76	51,46	0,8117	1,1459
10	Unigramas	TF-IDF	No	52,72	51,33	0,8112	1,1475
50	Unigramas	TF	Sí	52,45	51,11	0,8068	1,1677
50	Unigramas	TF	No	52,30	50,85	0,8066	1,1688
50	Unigramas	TF-IDF	Sí	50,24	48,90	0,7893	1,2114
50	Unigramas	TF-IDF	No	50,16	48,70	0,7887	1,2138

Al contrario que para el caso binario, en este se ve que funcionan mejor los valores de C más pequeños, quizá debido a que los datos de 5 clases se encuentran más dispersos y un hiperplano menos ajustado los diferencia mejor. Igualmente, funcionan mejor los unigramas y bigramas y, en este caso, el mejor experimento emplea un vector de ocurrencias, aunque hay diversidad en los resultados, por lo que no se puede concluir nada.

El mejor resultado consigue un 59,45 % de exactitud y, aunque el error es algo

superior al de bayes, se obtiene una buena (aunque no excelente) área bajo la curva, acercándose la exactitud al 60 %, un porcentaje más cercano al estado del arte y bastante superior al clasificador bayesiano.

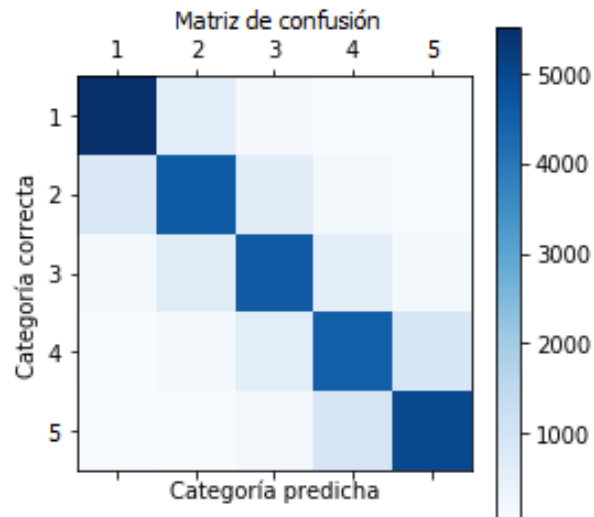


Fig. 5.4. Matriz de confusión de SVM multiclase

Observando la matriz de confusión vemos que el clasificador no tiene mucho problema en diferenciar las clases, y cuando se equivoca lo hace hacia las clases adyacentes. También queda claro que las estrellas que predice con más facilidad son las de los extremos, 1 y 5 estrellas.

5.2.3. Random Forest

A continuación, se muestran los resultados obtenidos con el método de *ensemble* Random Forest. Primeramente, en el conjunto binario:

TABLA 5.7. RESULTADOS DE RANDOM FOREST EN EL CONJUNTO BINARIO

Parámetros				Resultados			
Árboles	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
200	Unigramas	TF-IDF	No	82,92	82,92	0,9120	0,4828
200	Unigramas	TF-IDF	No	82,84	83,11	0,9113	0,4744
200	Unigramas	TF	Sí	82,75	83,08	0,9108	0,4850
200	Unigramas	TF	No	82,71	82,88	0,9098	0,4789

200	Uni. y bi.	TF	Sí	82,18	82,11	0,9047	0,5459
200	Uni. y bi.	TF-IDF	Sí	82,14	82,42	0,9044	0,5305
200	Uni. y bi.	TF-IDF	No	82,08	81,95	0,9034	0,5354
200	Uni. y bi.	TF	No	82,01	81,92	0,9031	0,5461
100	Unigramas	TF-IDF	Sí	82,00	81,86	0,9033	0,4853
100	Unigramas	TF	No	81,98	82,25	0,9034	0,4841
100	Unigramas	TF-IDF	Sí	81,88	82,20	0,9013	0,4873
100	Unigramas	TF	Sí	81,80	82,09	0,9008	0,4850
100	Uni. y bi.	TF	Sí	81,37	81,67	0,8981	0,5348
100	Uni. y bi.	TF-IDF	Sí	81,33	81,60	0,8986	0,5406
100	Uni. y bi.	TF-IDF	No	81,28	81,36	0,8973	0,5337
100	Uni. y bi.	TF	No	81,22	81,13	0,8966	0,5359
50	Unigramas	TF-IDF	No	81,22	80,93	0,8965	0,4933
50	Unigramas	TF-IDF	Sí	81,11	81,17	0,8947	0,4914
50	Unigramas	TF	No	80,99	81,16	0,8953	0,4916
50	Unigramas	TF	Sí	80,86	80,50	0,8945	0,4900
50	Uni. y bi.	TF	Sí	80,76	80,47	0,8928	0,5488
50	Uni. y bi.	TF	No	80,76	80,39	0,8933	0,5433
50	Uni. y bi.	TF-IDF	Sí	80,75	80,77	0,8931	0,5480
50	Uni. y bi.	TF-IDF	No	80,75	80,38	0,8938	0,5475

Sorprendentemente, los experimentos con Random Forest proporcionan peores resultados que el resto de modelos, con un mayor error, peor exactitud, peor f1 y peor área bajo la curva. Se detecta que el número de árboles es el factor principal que hace más efecto en el resultado, obteniendo mejores cuando se incrementan. Esto nos puede indicar que si se aumentase aún más el número de árboles, el resultado mejoraría o se acercaría a los de los otros modelos, pero tiene el punto negativo de que aumentar el número de árboles aumenta proporcionalmente el tiempo del entrenamiento, el cuál ya ha sido excesivamente superior al resto de modelos, por lo que se descarta.

Se observa además que, siendo el número de árboles el parámetro principal, dentro de cada número de árboles funcionan mejor todos los experimentos que utilizan únicamente unigramas, a diferencia del resto de modelos donde es al contrario. Esto

podría deberse a que al tomar las decisiones, es decir, realizar las ramificaciones del árbol, el proceso se realice mejor unitariamente, es decir, empleando unigramas.

TABLA 5.8. RESULTADOS DE RANDOM FOREST EN EL CONJUNTO MULTICLASE

Parámetros				Resultados			
Árboles	Ngramas	Vector	Sublin.1	Acc. ↑	F1	ROC-AUC	C-ent. loss
200	Unigramas	TF-IDF	No	52,40	50,85	0,7610	1,2816
200	Unigramas	TF	No	52,26	50,75	0,7628	1,2794
200	Unigramas	TF-IDF	Sí	52,24	50,70	0,7635	1,2809
200	Unigramas	TF	Sí	52,22	50,77	0,7691	1,2795
200	Uni. y bi.	TF	Sí	51,57	49,65	0,7635	1,3614
200	Uni. y bi.	TF	No	51,55	49,60	0,7631	1,3624
200	Uni. y bi.	TF-IDF	No	51,52	49,45	0,7532	1,3625
200	Uni. y bi.	TF-IDF	Sí	51,38	49,25	0,7546	1,3626
100	Unigramas	TF-IDF	Sí	51,25	49,75	0,7599	1,2842
100	Unigramas	TF	Sí	51,08	49,60	0,7589	1,2844
100	Unigramas	TF	No	51,01	49,51	0,7577	1,2836
100	Unigramas	TF-IDF	No	51,00	49,43	0,7622	1,2873
100	Uni. y bi.	TF-IDF	Sí	50,27	48,20	0,7523	1,3649
100	Uni. y bi.	TF	No	50,17	48,24	0,7523	1,3657
100	Uni. y bi.	TF-IDF	No	50,13	48,08	0,7503	1,3661
100	Uni. y bi.	TF	Sí	49,94	47,98	0,7494	1,3673
50	Unigramas	TF-IDF	No	49,29	47,84	0,7411	1,2964
50	Unigramas	TF	No	49,23	47,78	0,7402	1,2951
50	Unigramas	TF	Sí	49,12	47,76	0,7401	1,2966
50	Unigramas	TF-IDF	Sí	49,06	47,63	0,7508	1,2975
50	Uni. y bi.	TF-IDF	Sí	48,45	46,43	0,7555	1,3689
50	Uni. y bi.	TF	Sí	48,35	46,53	0,7537	1,3712
50	Uni. y bi.	TF	No	48,18	46,28	0,7546	1,3719
50	Uni. y bi.	TF-IDF	No	48,07	46,13	0,7549	1,3710

Los resultados son exactamente iguales que en el conjunto binario: son peores en todos los sentidos que el resto de algoritmos, más costosos de entrenar y sus resultados mejoran conforme se incrementa el número de árboles.

Mirando la matriz de confusión, vemos que el problema reside en lo mismo que en los otros clasificadores, al modelo le cuesta mucho diferenciar entre las estrellas centrales y clasifica erróneamente las adyacentes.

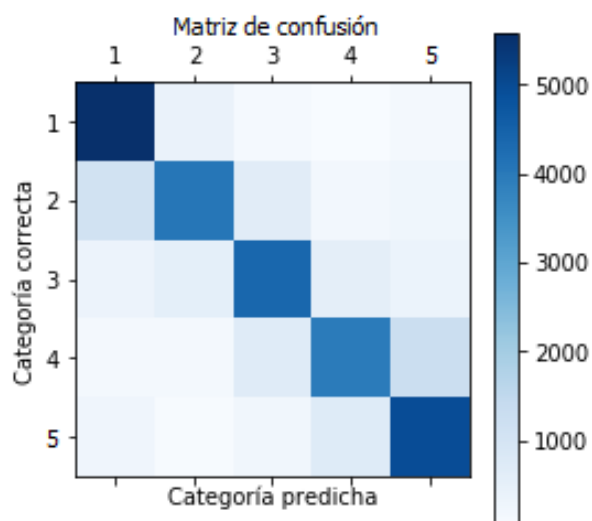


Fig. 5.5. Matriz de confusión de Random Forest multiclase

5.2.4. Regresión logística

En esta sección se describen los resultados de los experimentos realizados con el algoritmo de regresión logística y los parámetros definidos en la tabla 4.6. En primer lugar, se experimenta con el conjunto binario:

TABLA 5.9. RESULTADOS DE LA REGRESIÓN LOGÍSTICA EN EL CONJUNTO BINARIO

Parámetros				Resultados			
C	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
100	Uni. y bi.	TF-IDF	Sí	87,19	87,16	0,9450	0,3489
500	Uni. y bi.	TF-IDF	Sí	87,17	87,14	0,9449	0,4068
10	Uni. y bi.	TF-IDF	Sí	87,12	87,09	0,9449	0,3027
100	Uni. y bi.	TF-IDF	No	87,10	87,05	0,9442	0,3512
500	Uni. y bi.	TF-IDF	No	87,10	87,05	0,9440	0,4093
10	Uni. y bi.	TF-IDF	No	87,05	87,00	0,9441	0,3049
10	Uni. y bi.	TF	Sí	86,98	86,95	0,9441	0,3044

100	Uni. y bi.	TF	Sí	86,94	86,90	0,9434	0,3501
10	Uni. y bi.	TF	No	86,92	86,88	0,9435	0,3056
500	Uni. y bi.	TF	Sí	86,91	86,88	0,9429	0,4071
100	Uni. y bi.	TF	No	86,89	86,85	0,9427	0,3497
500	Uni. y bi.	TF	No	86,83	86,79	0,9421	0,4057
1	Uni. y bi.	TF-IDF	Sí	86,36	86,32	0,9396	0,3255
1	Uni. y bi.	TF-IDF	No	86,26	86,21	0,9387	0,3274
1	Uni. y bi.	TF	Sí	86,25	86,19	0,9389	0,3190
1	Unigramas	TF-IDF	Sí	86,23	86,19	0,9373	0,3236
1	Uni. y bi.	TF	No	86,16	86,10	0,9379	0,3216
1	Unigramas	TF-IDF	No	86,08	86,02	0,9363	0,3264
10	Unigramas	TF	Sí	85,96	85,92	0,9363	0,3232
1	Unigramas	TF	Sí	85,92	85,85	0,9359	0,3256
10	Unigramas	TF	No	85,89	85,84	0,9358	0,3242
1	Unigramas	TF	No	85,80	85,72	0,9349	0,3282
10	Unigramas	TF-IDF	Sí	85,61	85,58	0,9328	0,3364
10	Unigramas	TF-IDF	No	85,52	85,49	0,9322	0,3374
0,1	Unigramas	TF-IDF	Sí	84,88	84,86	0,9269	0,3913
100	Unigramas	TF	Sí	84,66	84,66	0,9245	0,3861
0,1	Unigramas	TF-IDF	No	84,66	84,63	0,9251	0,3948
100	Unigramas	TF	No	84,61	84,60	0,9242	0,3840
0,1	Unigramas	TF	Sí	84,17	84,09	0,9219	0,3812
0,1	Unigramas	TF	No	83,97	83,88	0,9200	0,3852
0,1	Uni. y bi.	TF-IDF	Sí	83,83	83,80	0,9196	0,4503
0,1	Uni. y bi.	TF	Sí	83,77	83,73	0,9186	0,3939
0,1	Uni. y bi.	TF-IDF	No	83,70	83,69	0,9179	0,4482
500	Unigramas	TF	No	83,70	83,71	0,9155	0,4557
500	Unigramas	TF	Sí	83,69	83,70	0,9159	0,4612
0,1	Uni. y bi.	TF	No	83,61	83,55	0,9171	0,3964
100	Unigramas	TF-IDF	Sí	83,46	83,47	0,9145	0,4590
100	Unigramas	TF-IDF	No	83,42	83,43	0,9139	0,4563
500	Unigramas	TF-IDF	Sí	82,22	82,25	0,9029	0,6011
500	Unigramas	TF-IDF	No	82,14	82,19	0,9022	0,5942
0,001	Uni. y bi.	TF-IDF	Sí	80,34	80,58	0,8883	0,6845
0,001	Unigramas	TF-IDF	Sí	80,00	80,43	0,8843	0,6667
0,001	Uni. y bi.	TF-IDF	No	79,90	80,10	0,8829	0,6837
0,001	Unigramas	TF-IDF	No	79,78	80,19	0,8814	0,6665

0,001	Unigramas	TF	Sí	77,54	77,50	0,8578	0,6435
0,001	Uni. y bi.	TF	Sí	77,32	77,21	0,8562	0,6607
0,001	Unigramas	TF	No	77,24	77,03	0,8534	0,6434
0,001	Uni. y bi.	TF	No	77,00	76,84	85,09	0,6588

Los experimentos obtienen muy buenos resultados en comparación con el resto de modelos, alcanzando una exactitud de 87,19%. Al ser modelo basado en una función, de la misma forma que SVM, los parámetros funcionan de forma similar. Por un lado, los valores de C que mejores resultados ofrecen son los más altos (500, 10, 100), porque son los que hacen que la función se ajuste más a los datos. También, la vectorización que mejor funciona es la de frecuencia inversa y la utilización de unigramas y bigramas.

Lo mismo ocurre con los resultados, son muy similares: un excelente resultado del área bajo la curva, muy cercano a 1, buena exactitud y f1 y un error bajo aunque no tan bajo como bayes. Respecto a la matriz de confusión, de nuevo es muy similar, sin problemas para diferenciar ambas polaridades:

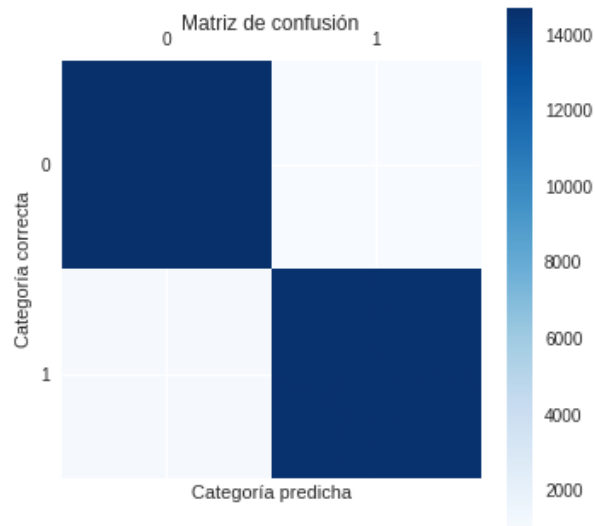


Fig. 5.6. Matriz de confusión de la regresión logística binaria

**TABLA 5.10. RESULTADOS DE LA REGRESIÓN
LOGÍSTICA EN EL CONJUNTO MULTICLASE**

Parámetros				Resultados			
C	Ngramas	Vector	Sublin.	Acc. ↑	F1	ROC-AUC	C-ent. loss
10	Uni. y bi.	TF-IDF	Sí	59,25	58,92	0,8605	0,9619
1	Uni. y bi.	TF-IDF	Sí	59,14	58,77	0,8634	1,0068
1	Uni. y bi.	TF	Sí	59,05	58,62	0,8631	0,9878
10	Uni. y bi.	TF-IDF	No	59,05	58,71	0,8697	0,9666
1	Uni. y bi.	TF-IDF	No	58,94	58,55	0,8725	1,0094
10	Uni. y bi.	TF	Sí	58,92	58,56	0,8722	0,9681
10	Uni. y bi.	TF	No	58,81	58,45	0,8599	0,9709
1	Uni. y bi.	TF	No	58,81	58,36	0,8636	0,9927
100	Uni. y bi.	TF-IDF	Sí	58,57	58,23	0,8686	1,1021
100	Uni. y bi.	TF-IDF	No	58,42	58,08	0,8680	1,1093
500	Uni. y bi.	TF-IDF	Sí	58,39	58,05	0,8673	1,2758
500	Uni. y bi.	TF-IDF	No	58,22	57,86	0,8700	1,2850
1	Unigramas	TF-IDF	Sí	58,05	57,55	0,8687	1,0047
1	Unigramas	TF	Sí	57,98	57,45	0,8694	1,0031
100	Uni. y bi.	TF	Sí	57,93	57,58	0,8670	1,1208
1	Unigramas	TF-IDF	No	57,92	57,41	0,8692	1,0097
100	Uni. y bi.	TF	No	57,85	57,50	0,8698	1,1206
1	Unigramas	TF	No	57,75	57,21	0,8684	1,0079
500	Uni. y bi.	TF	Sí	57,65	57,30	0,8688	1,3068
500	Uni. y bi.	TF	No	57,49	57,15	0,8797	1,3058
0,1	Unigramas	TF-IDF	Sí	56,98	56,28	0,8721	1,1306
10	Unigramas	TF	Sí	56,78	56,36	0,8570	1,0168
0,1	Unigramas	TF-IDF	No	56,71	55,99	0,8565	1,1361
10	Unigramas	TF	No	56,71	56,30	0,8629	1,0179
0,1	Unigramas	TF	Sí	56,22	55,50	0,8556	1,1049
0,1	Uni. y bi.	TF-IDF	Sí	56,08	55,41	0,8552	1,2418
0,1	Unigramas	TF	No	56,00	55,26	0,8677	1,1116
0,1	Uni. y bi.	TF	Sí	55,94	55,27	0,8696	1,1294
0,1	Uni. y bi.	TF-IDF	No	55,75	55,07	0,8683	1,2365
0,1	Uni. y bi.	TF	No	55,73	55,05	0,8539	1,1329
10	Unigramas	TF-IDF	Sí	55,63	55,28	0,8538	1,0614
10	Unigramas	TF-IDF	No	55,43	55,09	0,8582	1,0623
100	Unigramas	TF	Sí	53,72	53,45	0,8286	1,2147

100	Unigramas	TF	No	53,67	53,41	0,8369	1,2067
0,001	Uni. y bi.	TF-IDF	Sí	52,40	51,01	0,8168	1,5993
500	Unigramas	TF	Sí	52,01	51,82	0,8146	1,4407
500	Unigramas	TF	No	51,96	51,78	0,8161	1,4213
0,001	Uni. y bi.	TF-IDF	No	51,60	50,04	0,8058	1,5984
100	Unigramas	TF-IDF	Sí	51,38	51,21	0,8028	1,4451
100	Unigramas	TF-IDF	No	51,33	51,17	0,8068	1,4339
0,001	Unigramas	TF-IDF	Sí	51,16	49,27	0,8068	1,5780
0,001	Unigramas	TF-IDF	No	50,77	48,90	0,8015	1,5775
500	Unigramas	TF-IDF	No	49,33	49,23	0,7798	1,8471
500	Unigramas	TF-IDF	Sí	49,32	49,22	0,7808	1,8728
0,001	Uni. y bi.	TF	Sí	47,40	45,38	0,7512	1,5701
0,001	Unigramas	TF	Sí	47,12	44,92	0,7471	1,5475
0,001	Uni. y bi.	TF	No	46,47	44,40	0,7371	1,5674
0,001	Unigramas	TF	No	46,39	44,16	0,7361	1,5469

Con los resultados de la experimentación multiclase ocurre lo mismo que anteriormente, se observan las mismas tendencias que en las máquinas de soporte vectorial: los unigramas y bigramas y la vectorización en TF-IDF dan resultados los valores de C inferiores obtienen mejores resultados posiblemente porque al ajustarse menos a los datos aproximan mejor los datos de las 5 estrellas, que se encuentran más dispersos. Igualmente, obtiene buenos resultados, acercándose al 60 % de exactitud y con un error similar (aunque algo superior).

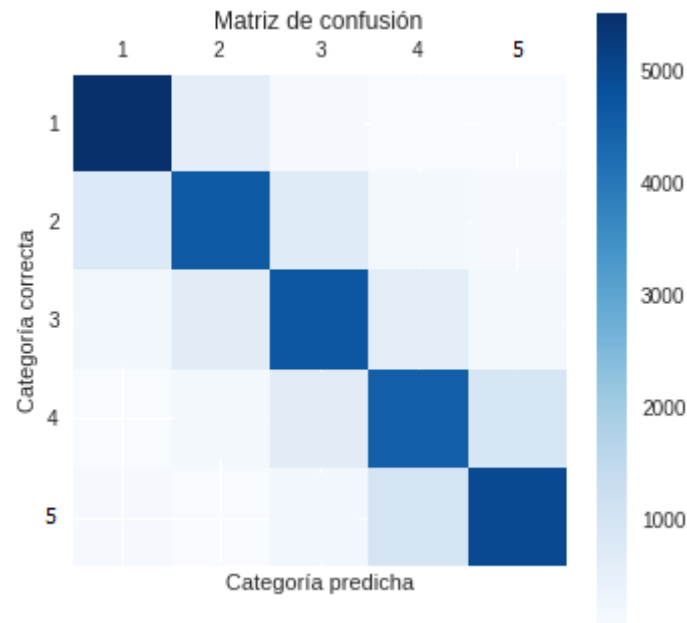


Fig. 5.7. Matriz de confusión de la regresión logística multiclase

En la matriz de confusión podemos ver lo mismo que en los demás experimentos multiclase, le cuesta más predecir las estrellas centrales que las de los extremos.

5.2.5. Comparativa general de bag-of-words

A continuación, tras la realización de toda la experimentación con los algoritmos con los que se prueba el modelo *bag-of-words*, se procede a realizar una comparativa de los distintos algoritmos, que de resultado al modelo final de *bag-of-words* para el problema.

Los mejores resultados de cada algoritmo se pueden encontrar en las siguientes tablas:

**TABLA 5.11. RESULTADOS DE BAG-OF-WORDS EN
CONJUNTO BINARIO**

	Accuracy	F1	ROC-AUC	Cross-entropy loss
Naïve Bayes	83,99	83,61	0,9194	0,1675
SVM	87,22	87,19	0,9452	0,3008
Random Forest	82,92	82,92	0,9120	0,4828
Regresión log.	87,19	87,16	0,9450	0,3489

**TABLA 5.12. RESULTADOS DE BAG-OF-WORDS EN
CONJUNTO MULTICLASE**

	Accuracy	F1	ROC-AUC	Cross-entropy loss
Naïve Bayes	55,30	55,57	0,8053	1,0767
SVM	59,65	59,25	0,8652	0,9658
Random Forest	52,40	50,85	0,7610	1,2816
Regresión log.	59,25	58,92	0,8605	0,9619

Aunque se pueden ver más visualmente en la figura 5.8.

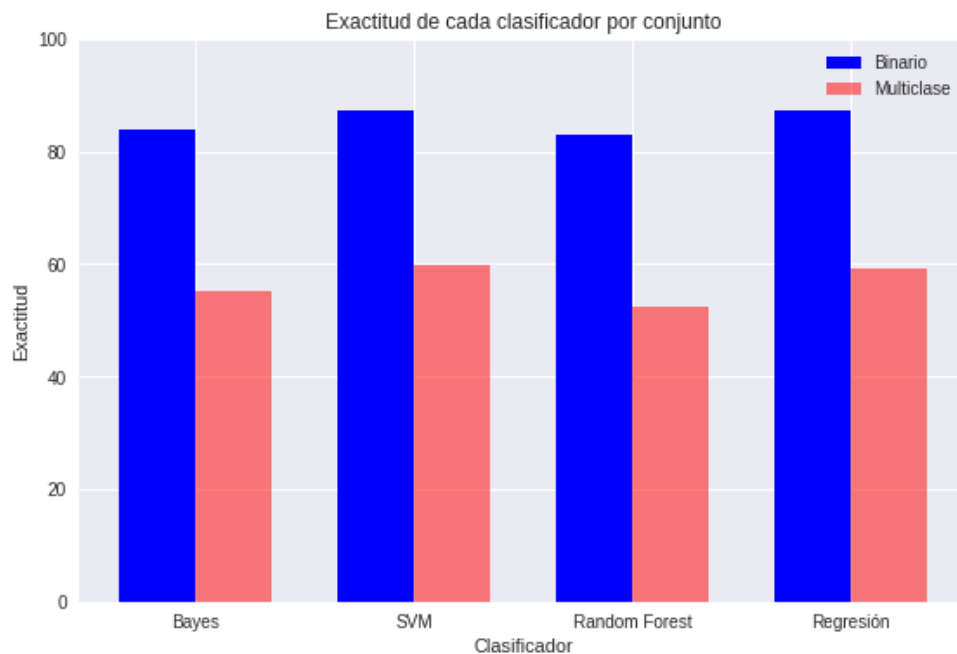


Fig. 5.8. Comparativa de resultados de bag-of-words

El algoritmo que mejores resultados ha obtenido ha sido la máquina de soporte vectorial (SVM), aunque también se debe tener en cuenta la regresión logística, que ha obtenido resultados muy similares. La posible razón detrás de la diferencia entre estos dos algoritmos y Random Forest y bayes ingenuo es que estos dos últimos están basados en lógica y probabilidad, respectivamente, mientras que los otros están basados en funciones analíticas, lo cuál podría provocar que aproximen de mejor manera los datos de nuestro problema.

Como conclusiones que se pueden sacar del propio modelo *bag-of-words*, se ha visto que, en casi todos los casos menos en los árboles de decisión, los mejores resultados los ha proporcionado la utilización de unigramas y bigramas. Esto se debe a que el empleo de bigramas proporciona mayor contexto a las palabras y el modelo consigue predecir mejor las reseñas. Por otro lado, aunque se han obtenido resultados variados respecto a la vectorización, los mejores resultados en todos los casos han sido mediante la frecuencia inversa, debido a que penaliza aquellas palabras que aparecen demasiado en el texto y por tanto no aportan valor a la predicción. No se ha podido sacar ninguna conclusión acerca de la modificación sublineal de la frecuencia dado que se ha producido diversidad en los resultados.

Respecto a los conjuntos, en el binario, al tener únicamente que clasificar dos polaridades, se han conseguido muy buenos modelos que se acercan al 90 % de exactitud, pero la tarea es más complicada al predecir las 5 estrellas porque no existe gran diferencia entre las reseñas que han sido valoradas con algunas de las estrellas centrales (2, 3 o 4), lo que genera dificultades para el modelo. Aún así, se han obtenido buenos resultados, cercanos al 60 %, teniendo en cuenta que si dejásemos el problema al puro azar las probabilidades de acertar serían del 20 %.

5.3. Evaluación de los modelos con word embeddings

A continuación, se representan todos los resultados obtenidos en ambos conjuntos al experimentar con las dos arquitecturas de redes profundas con *word embeddings* planteadas.

Los resultados de todos los experimentos se representan en el orden en el que se han

realizado, para que se pueda ver de alguna manera la toma de decisiones llevada a cabo. Además, en todos los casos se han tenido en cuenta las 20000 palabras más relevantes del total del vocabulario de las reseñas para construir los vectores con los *word embeddings* preentrenados, pues tener en cuenta todo el vocabulario completo resulta inviable. También, se considera como parámetro a experimentar el entrenar o no los word embeddings. Esto es, cuando los vectores preentrenados se introducen en la primera capa para multiplicar los pesos de las palabras por las palabras de entrada, éstas últimas toman los pesos que tienen los vectores preentrenados. No obstante, se pueden entrenar estas capas para que los pesos se ajusten al texto de entrada.

Los ciclos que se ha entrenado cada experimento ha sido hasta que el error de la red ha comenzado a aumentar.

En primer lugar, se muestran los resultados obtenidos con la red neuronal convolucional en forma piramidal de Johnson y Zhang (2017).

5.3.1. Red neuronal convolucional profunda en forma piramidal

Partiendo de la arquitectura planteada por los autores y descrita en el capítulo anterior, se comenzará también por los parámetros definidos por Johnson y Zhang (2017), que indican que emplean el optimizador SGD y un decremento en la tasa de aprendizaje.

TABLA 5.13. RESULTADOS DE DPCNN EN EL CONJUNTO BINARIO

Parámetros							Resultados			
Entrenar embeddings	Entrada	Optimizador	Filtros	Tasa	Decremento	Ciclos	Acc.	F1	ROC-AUC	Loss
No	250	SGD	64	0,1	0,000001	4	86,10	85,41	0,9427	0,3363
No	250	SGD	64	0,1	0,00001	4	85,92	85,10	0,9416	0,3555
No	250	SGD	64	0,1	0,0001	4	86,64	86,53	0,9424	0,3313
No	250	SGD	64	0,1	0,001	6	86,16	86,12	0,9390	0,3823
No	250	SGD	64	0,1	-	2	85,77	85,83	0,9362	0,3389
No	250	SGD	64	0,01	0,001	2	85,89	85,68	0,9374	0,3795
No	250	SGD	64	0,01	0,0001	1	85,91	85,64	0,9359	0,3789
No	250	SGD	64	0,01	0,00001	1	85,70	85,32	0,9353	0,4074
No	250	SGD	64	0,01	0,01	4	85,98	85,83	0,9382	0,3426
No	250	SGD	64	0,01	-	2	86,31	85,84	0,9416	0,3340
No	250	SGD	64	1	0,0001	1	49,99	66,66	0,5000	8,0605
No	250	SGD	64	1	0,1	1	53,81	61,29	0,5490	2,6311
No	250	SGD	64	1	-	1	49,99	66,66	0,5000	8,0605
No	250	SGD	64	0,001	0,0001	8	84,45	84,41	0,9246	0,3597
No	250	SGD	64	0,001	0,001	6	82,98	83,00	0,9090	0,3939
No	250	SGD	64	0,05	-	2	84,08	85,16	0,9356	0,3636
No	250	SGD	64	0,05	0,0001	2	85,51	85,56	0,9355	0,3386
No	250	SGD	64	0,05	0,000001	3	84,89	83,62	0,9375	0,3664
No	250	SGD	64	0,05	0,01	1	86,29	86,06	0,9400	0,3254

No	250	SGD	64	0,05	0,005	1	86,46	86,28	0,9410	0,3250
No	250	SGD	250	0,05	0,01	4	85,56	85,81	0,9352	0,3688
No	500	SGD	250	0,05	0,005	4	85,69	85,05	0,9378	0,3614
No	500	SGD	64	0,1	0,005	3	85,44	84,91	0,9331	0,3450
No	500	SGD	64	0,1	0,01	3	84,77	84,48	0,9284	0,3526
No	500	SGD	64	0,05	0,005	7	85,59	85,22	0,9349	0,3357
No	500	SGD	64	0,05	0,00001	1	85,08	85,04	0,9308	0,3513
No	500	SGD	64	0,05	0,000001	1	84,36	83,18	0,9315	0,3566
No	500	SGD	64	0,05	0,0001	4	85,96	85,21	0,9402	0,3300
No	500	SGD	64	0,05	0,0005	5	86,33	85,91	0,9405	0,3229
No	500	SGD	64	0,005	0,0001	2	83,92	82,56	0,9288	0,3683
Sí	500	SGD	64	0,05	0,005	4	86,18	86,18	0,9396	0,3256
Sí	500	SGD	64	0,05	0,01	5	85,59	85,77	0,9353	0,3353
Sí	500	SGD	64	0,05	0,001	4	87,01	87,04	0,9451	0,3135
Sí	500	SGD	64	0,05	-	6	87,29	86,83	0,9477	0,3105
Sí	500	SGD	64	0,05	0,0001	87,30	87,08	0,9473	0,3081	0,3081
Sí	500	SGD	64	0,1	-	4	87,08	86,50	0,9469	0,3182
Sí	500	SGD	64	0,1	0,0001	3	87,29	87,10	0,9473	0,3086
Sí	500	SGD	64	0,1	0,01	6	86,06	85,85	0,9388	0,3270
Sí	500	SGD	64	0,1	0,001	2	86,92	87,17	0,9459	0,3184
Sí	500	SGD	64	0,1	0,0005	4	87,35	87,06	0,9477	0,3141
Sí	500	SGD	250	0,1	0,0005	4	87,11	87,37	0,9479	0,3451

Sí	500	SGD	250	0,05	-	4	87,22	87,37	0,9480	0,3436
Sí	500	SGD	250	0,05	0,0001	3	87,07	86,57	0,9470	0,3344
Sí	500	Adam	64	0,05	0,0001	2	84,32	85,17	0,9302	0,4088
Sí	500	Adam	64	0,05	-	3	84,69	84,24	0,9267	0,4474
Sí	500	Adam	64	0,1	-	3	84,68	85,06	0,9288	0,4255
Sí	500	Adam	64	0,001	-	2	86,94	86,50	0,9458	0,3096
Sí	500	Adam	64	0,01	-	1	85,06	83,73	0,9391	0,3727
Sí	500	Adam	64	0,001	0,0001	3	86,82	86,87	0,9441	0,3137
No	500	Adam	64	0,001	0,0001	3	86,01	86,24	0,9402	0,3248
Sí	500	Adam	64	0,001	0,000005	2	86,88	86,76	0,9446	0,3108
Sí	500	Adam	64	0,01	0,0001	1	86,73	86,44	0,9414	0,3293
Sí	500	Adam	64	0,01	0,005	2	86,89	86,40	0,9445	0,3313
Sí	500	Adam	64	0,01	0,01	2	85,20	85,39	0,9311	0,4245

En un principio, la arquitectura planteada cuenta con 250 filtros convolucionales, pero se ha probado con 64 dado que se reducía el tiempo de cómputo y, además, se ha comprobado que aumentar el número de filtros no mejoraba los resultados. Lo que sí se ha visto que ha mejorado los resultados ha sido, como era de esperar, aumentar el número de palabras de entrada de 250 a 500, de forma que la red tomase más palabras en cuenta y el tamaño de las entradas se asemeje más a la media de la longitud de las reseñas (tabla 3.2).

También ha funcionado mejor entrenar los pesos de los *embeddings* que dejarlos congelados, y se ha probado con el optimizador Adam pero no se han obtenido mejores resultados que con SGD.

En general, se han obtenido resultados consistentes con el modelo y los valores de F1 y el área bajo la curva demuestran que son buenos modelos. Viendo la matriz de confusión, se resalta este resultado y se ve claramente que el modelo no tiene problemas en diferenciar las polaridades.

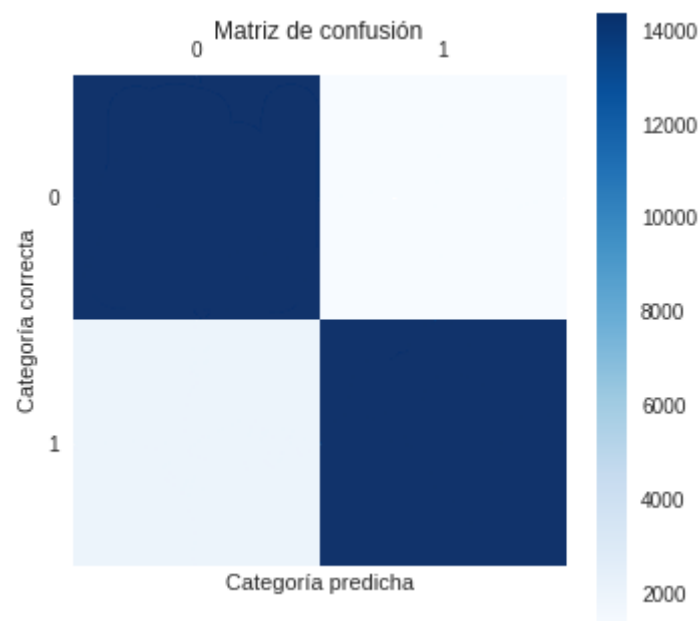


Fig. 5.9. Matriz de confusión de DPCNN en el conjunto binario

A continuación, se muestran los resultados en el conjunto multiclase:

TABLA 5.14. RESULTADOS DE DPCNN EN EL CONJUNTO MULTICLASE

Parámetros							Resultados			
Entrenar embeddings	Entrada	Optimizador	Filtros	Tasa	Decremento	Ciclos	Acc.	F1	ROC-AUC	Loss
Sí	500	SGD	64	0,1	0,0005	4	60,26	61,80	0,8809	0,9218
Sí	500	SGD	64	0,1	-	4	60,10	60,87	0,8792	0,9265
Sí	500	SGD	64	0,05	-	5	60,39	60,53	0,8797	0,9327
Sí	500	SGD	64	0,01	-	6	60,45	62,77	0,8836	0,9150
Sí	500	SGD	64	0,001	-	6	56,54	59,62	0,8598	0,9996
Sí	500	SGD	64	0,05	0,0001	2	59,30	61,76	0,8761	0,9526
Sí	500	SGD	64	0,01	0,00001	4	59,29	62,05	0,8778	0,9394
Sí	500	SGD	64	0,5	-	2	51,16	54,05	0,8285	1,1216
Sí	500	SGD	64	0,5	0,0001	4	57,48	59,85	0,8717	1,0297
Sí	500	SGD	64	0,5	0,005	5	57,55	60,53	0,8698	1,0126
Sí	500	SGD	64	0,5	0,05	6	36,01	47,51	0,7453	1,2540
Sí	500	SGD	64	0,1	0,005	6	58,92	61,33	0,8756	0,9506
Sí	500	SGD	64	0,1	0,00001	3	60,35	59,94	0,8776	0,9393
Sí	500	SGD	64	0,1	0,000001	6	60,60	59,15	0,8824	0,9237
Sí	500	SGD	64	0,1	0,000005	8	60,47	58,33	0,8788	0,9642
Sí	500	SGD	64	0,1	0,0000001	5	59,69	62,05	0,8788	0,9421
Sí	500	SGD	64	0,05	0,00001	4	60,12	62,87	0,8828	0,9215
Sí	500	SGD	64	0,05	0,000001	5	60,21	61,82	0,8781	0,9283
Sí	500	SGD	64	0,01	0,005	8	58,36	61,06	0,8712	0,9629

Sí	500	SGD	64	0,01	0,0001	6	60,36	62,70	0,8830	0,9242
Sí	500	SGD	64	0,01	0,000005	3	59,97	62,06	0,8801	0,9372
Sí	500	SGD	250	0,1	0,000001	3	60,54	54,49	0,8793	0,9536
Sí	500	SGD	250	0,1	-	5	61,03	60,49	0,8811	0,9368
Sí	500	SGD	250	0,05	-	4	60,22	61,25	0,8798	0,9609
Sí	500	SGD	250	0,01	-	6	60,55	62,83	0,8825	0,9559
No	500	SGD	250	0,1	-	4	58,99	60,29	0,8690	0,9949
No	500	SGD	250	0,05	-	6	59,30	59,98	0,8750	0,9729
No	500	SGD	250	0,01	-	4	57,94	61,21	0,8715	0,9991
Sí	500	Adam	64	0,1	-	2	19,90	33,33	0,5000	1,6095
Sí	500	Adam	64	0,01	-	1	57,08	57,89	0,8682	1,0440
Sí	500	Adam	64	0,001	-	3	59,59	61,09	0,8806	0,9649
Sí	500	Adam	64	0,0001	-	3	58,07	60,13	0,8697	0,9706
Sí	500	Adam	64	0,001	0,000001	3	59,65	60,72	0,8805	0,9586
Sí	500	Adam	64	0,001	0,0001	2	58,64	61,48	0,8770	0,9673
Sí	500	Adam	64	0,01	0,0001	2	59,00	58,44	0,8725	0,9963
Sí	500	Adam	64	0,05	0,01	3	59,28	54,03	0,8738	0,9965
Sí	500	Adam	64	0,01	0,1	10	58,11	60,33	0,8682	0,9670
Sí	500	Adam	64	0,01	0,001	2	58,83	59,01	0,8768	0,9754
Sí	500	Adam	64	0,01	0,01	2	60,44	62,79	0,8833	0,9313

Se han considerado las tendencias vistas al aplicar el modelo en el conjunto binario, comenzando con 500 neuronas de entrada y entrenando los vectores de los *embeddings*. Como era posible que no entrenar los embeddings funcionase bien, se ha probado esta posibilidad, pero ha arrojado el mismo resultado que en el conjunto anterior: funciona mejor entrenando esta capa.

De nuevo, se observan resultados consistentes y viendo la matriz de confusión se ve, como anteriormente, que las estrellas que más dificultad tiene el modelo para predecir son las centrales.

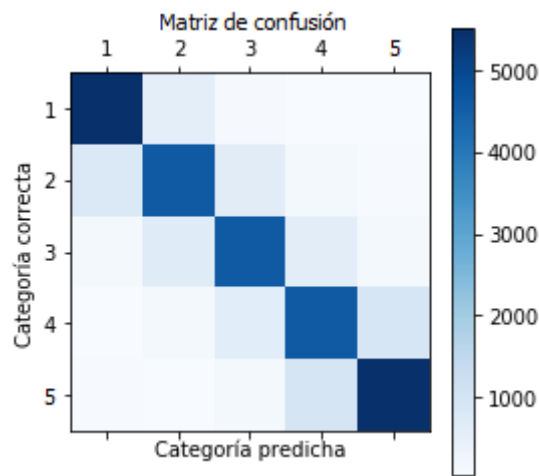


Fig. 5.10. Matriz de confusión de DPCNN
en el conjunto multiclase

5.3.2. LSTM bidireccional con reducción en dos dimensiones

En esta sección se muestran los resultados obtenidos con la arquitectura BLSTM descrita en Zhou et al. (2016). El número de experimentos con esta arquitectura es menor que con la anterior debido a que la arquitectura convolucional con capas de reducción reducía tanto el espacio que el entrenamiento era rápido, mientras que al emplear muchas neuronas LSTM el entrenamiento requiere más tiempo. Se ha comenzado probando con el optimizador Adadelta debido a que es el que mencionan los autores que mejores resultados ha dado con los parámetros utilizados.

Los resultados obtenidos en el conjunto binario son los siguientes:

TABLA 5.15. RESULTADOS DE BLSTM EN EL CONJUNTO BINARIO

Parámetros								Resultados			
Entrenar embed.	Entrada	Optimizador	Neuronas LSTM	Filtros	Tasa	Decremento	Ciclos	Acc.	F1	ROC-AUC	Loss
No	250	AdaDelta	300	100	1,0	-	3	85,33	85,30	0,9324	0,3485
Sí	250	AdaDelta	300	100	1,0	-	5	87,67	87,67	0,9493	0,2903
Sí	500	AdaDelta	300	100	1,0	-	7	87,71	87,70	0,9500	0,2896
Sí	500	AdaDelta	150	100	1,0	-	7	88,00	88,00	0,9514	0,2924
Sí	500	AdaDelta	150	50	1,0	-	8	87,70	87,70	0,9505	0,2924
Sí	500	AdaDelta	150-GRU	50	1,0	-	4	87,34	87,33	0,9486	0,2970
Sí	500	AdaDelta	300-GRU	50	1,0	-	5	87,64	87,64	0,9489	0,2922
Sí	500	AdaDelta	150-GRU	100	1,0	-	4	87,52	87,51	0,9486	0,3118
Sí	500	SGD	150	100	0,01	-	7	86,45	86,42	0,9426	0,3145
Sí	500	SGD	150	100	0,1	-	1	49,71	33,20	0,5000	8,1058
Sí	500	SGD	150	100	0,05	0,01	7	85,03	85,02	0,9312	0,3359
Sí	500	SGD	150	100	0,05	0,001	5	86,61	86,61	0,9434	0,3108
Sí	500	SGD	150	100	0,05	0,00001	1	49,71	33,20	0,5000	8,1058
Sí	500	SGD	150	100	0,05	0,1	4	82,41	82,41	0,9079	0,3888
Sí	500	SGD	150	100	0,01	0,0001	2	84,98	84,98	0,9299	0,3416

De acuerdo a Zhou et al. (2016), los mejores resultados se han obtenido con el optimizador AdaDelta, pero expresan que sería posible obtener mejores resultados utilizando diferente número de neuronas en la capa de LSTM. Por ello, se ha probado a cambiar las neuronas y, curiosamente, se han obtenido mejores resultados con la mitad de neuronas LSTM. También se ha probado a cambiar las neuronas LSTM por neuronas GRU (*Gated Recurrent Unit*), pero no se han mejorado los resultados y, posteriormente, se ha probado a cambiar de optimizador, con el mismo desenlace.

Se ha logrado un buen modelo alcanzando el 88 % de exactitud y, en general, se han obtenido buenos resultados, teniendo en cuenta los valores de F1 y del área bajo la curva, y muy similares a los obtenidos con la otra arquitectura aunque ligeramente mejores. Si observamos la matriz de confusión, vemos que ésta también es muy similar y que se clasifican sin problema las reseñas negativas y positivas.

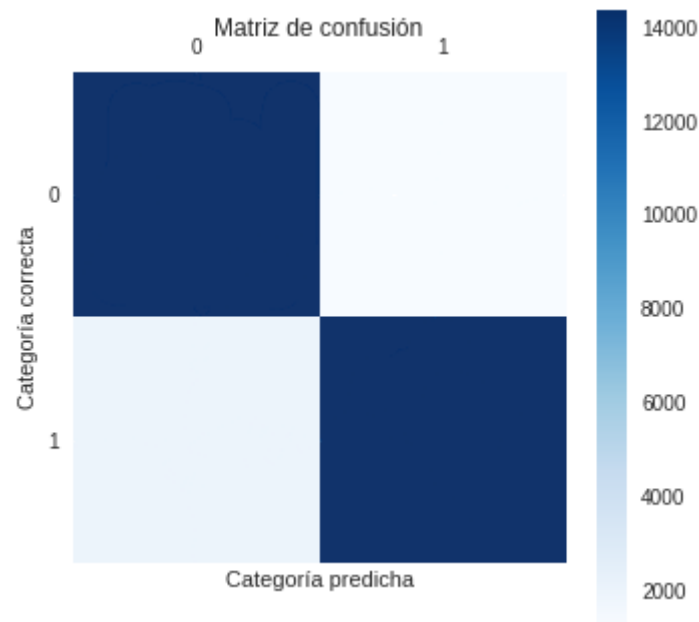


Fig. 5.11. Matriz de confusión de BLSTM en el conjunto binario

Siguiendo la misma idea de cambiar el número de neuronas pero probando el optimizador AdaDelta se han conseguido los siguientes resultados en el conjunto multiclase:

TABLA 5.16. RESULTADOS DE BLSTM EN EL CONJUNTO MULTICLASE

Parámetros								Resultados			
Entrenar embed.	Entrada	Optimizador	Neuronas LSTM	Filtros	Tasa	Decremento	Ciclos	Acc.	F1	ROC-AUC	Loss
No	250	AdaDelta	300	100	1,0	-	3	56,76	54,02	0,8256	1,0596
Sí	250	AdaDelta	300	100	1,0	-	5	60,85	58,37	0,8627	0,9255
Sí	500	AdaDelta	300	100	1,0	-	9	60,76	52,41	0,8578	0,9205
Sí	500	AdaDelta	150	100	1,0	-	8	61,33	58,44	0,8419	0,9023
Sí	500	AdaDelta	150	50	1,0	-	8	61,33	58,07	0,8505	0,9391
Sí	500	AdaDelta	150-GRU	50	1,0	-	4	61,27	57,83	0,8632	0,9142
Sí	500	AdaDelta	150-GRU	100	1,0	-	7	61,68	60,25	0,8704	0,9025
Sí	500	AdaDelta	300-GRU	50	1,0	-	8	61,33	58,69	0,8585	0,9087
Sí	500	AdaDelta	300-GRU	100	1,0	-	4	60,97	59,64	0,8626	0,9322
Sí	500	SGD	150-GRU	100	0,01	-	2	55,49	55,57	0,8290	1,0629
Sí	500	SGD	150-GRU	100	0,05	-	1	19,83	33,33	0,5096	8,0415
Sí	500	SGD	150-GRU	100	0,001	-	7	55,70	47,19	0,8260	1,0108
Sí	500	SGD	150-GRU	100	0,01	0,0001	2	56,51	41,10	0,8306	0,9956
Sí	500	SGD	150-GRU	100	0,01	0,001	8	57,67	47,12	0,8251	0,9728
Sí	500	SGD	150-GRU	100	0,01	0,1	5	51,40	49,58	0,7986	1,1194

Al igual que en el conjunto binario, se ha dado el resultado inesperado de que reduciendo el número de neuronas se han obtenido mejores resultados y, además, se han conseguido mejorar utilizando neuronas GRU en lugar de neuronas LSTM.

De nuevo, los resultados han sido similares a los obtenidos con la otra arquitectura aunque algo mejores y, observando los valores de F1 parecidos a la exactitud y los valores del área bajo la curva cercanos al 0,90 se puede concluir que, en general, se han conseguido buenos modelos. Viendo la matriz de confusión del mejor experimento llegamos a la misma conclusión.

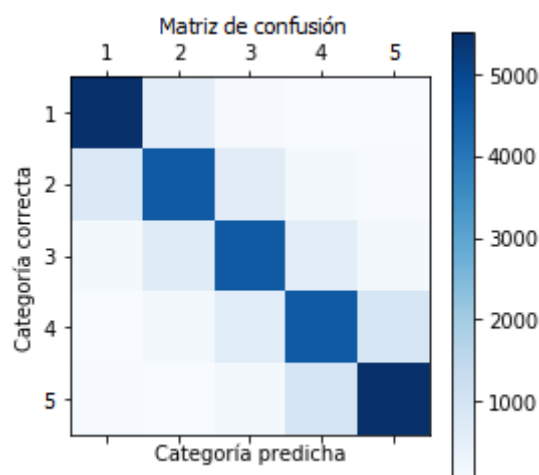


Fig. 5.12. Matriz de confusión de BLSTM
en el conjunto multiclase

5.4. Evaluación de ULMFiT

En esta sección se presentan los resultados obtenidos con el modelo ULMFiT, tanto para el conjunto binario como para el conjunto multiclase, igual que se ha hecho para el resto de modelos. No obstante, dada la complejidad del entrenamiento del modelo, no se representarán en una tabla todos los resultados obtenidos, sino que se expondrá el mejor experimento.

La complejidad mencionada se explica por cómo se entrena el modelo, lo cuál ya se ha descrito en el capítulo 4: en primer lugar, se entrena el modelo con los pesos preentrenados; después, se entrena el clasificador pero utilizando el ajuste discriminatorio y la descongelación de capas, es decir, a cada capa en cada ciclo se le aplica

una tasa de aprendizaje y se congelan o descongelan algunas capas.

En base a esto, se ha realizado la experimentación con el conjunto binario y se ha logrado el siguiente resultado:

TABLA 5.17. MEJOR RESULTADO DE ULMFIT EN EL
CONJUNTO BINARIO

	Accuracy	F1	ROC-AUC	Cross-entropy loss
ULMFiT	89,72	89,52	0,9647	0,2680

Para el ajuste del modelo preentrenado a los vectores de nuestras reseñas, primero se ha entrenado el modelo con las primeras capas congeladas en un ciclo de aprendizaje con tasa de aprendizaje 0,01 para que se mantengan las representaciones de palabras ya aprendidas en el modelo preentrenado. Después, se han descongelado todas las capas y se ha entrenado la red entera en 4 ciclos de aprendizaje con una tasa de 0,005.

Una vez se tiene el modelo ajustado, se pasa a entrenar la red de clasificación. Se ha comenzado con todas las capas congeladas, un primer ciclo de aprendizaje con un tasa de 0,01 y otro ciclo con una tasa de 0,05. Tras esto, se han descongelado las dos últimas capas y se ha entrenado un ciclo con tasa de aprendizaje 0,01. Se ha descongelado la siguiente capa y se ha entrenado con una tasa de 0,005. Para terminar, se ha descongelado la red completamente y se han ejecutado 4 ciclos de aprendizaje con una tasa de 0,001.

El resultado conseguido de un 89,72 % de exactitud se considera muy bueno y se puede valorar de buena manera el modelo teniendo en cuenta un área bajo la curva de 0,96 y el valor F1 muy cercano a la exactitud, lo que indica que el modelo tiene tanto buena precisión como buena exhaustividad. Además, el error que comete la red es bastante reducido y, viendo la matriz de confusión de la figura 5.13, se observa que no tiene problema en clasificar las reseñas positivas y negativas.

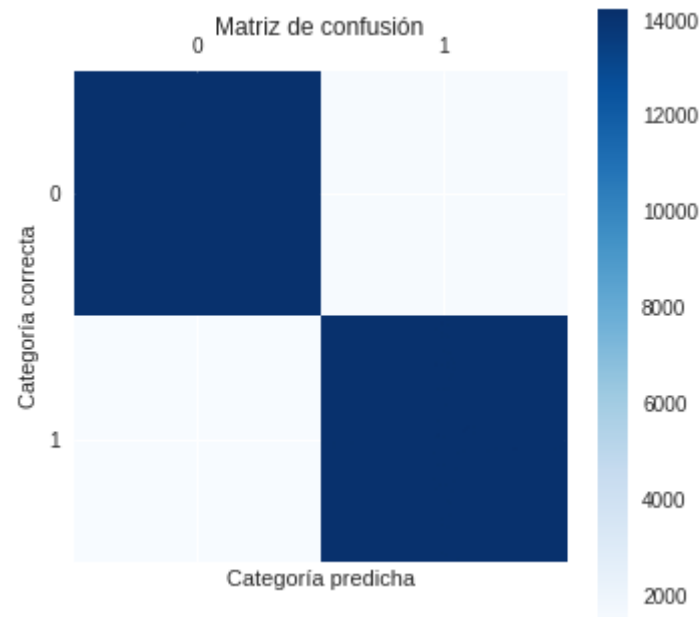


Fig. 5.13. Matriz de confusión de ULMFiT en el conjunto binario

Partiendo de los resultados obtenidos en la experimentación con el conjunto binario se ha pasado a emplear el modelo ULMFiT en el conjunto multiclase, de tal manera que el mejor resultado ha sido el siguiente:

TABLA 5.18. MEJOR RESULTADO DE ULMFiT EN EL CONJUNTO MULTICLASE

	Accuracy	F1	ROC-AUC	Cross-entropy loss
ULMFiT	63,67	62,98	0,8859	0,8753

El ajuste del modelo preentrenado ha sido el mismo que en el caso del conjunto binario: primero se ha entrenado el modelo con las primeras capas congeladas en un ciclo de aprendizaje con tasa de aprendizaje 0,01 para que se mantengan las representaciones de palabras ya aprendidas en el modelo preentrenado; después, se han descongelado todas las capas y se ha entrenado la red entera en 4 ciclos de aprendizaje con una tasa de 0,005.

Para el entrenamiento del clasificador ha funcionado especialmente bien el entrenamiento discriminatorio. Con la red congelada, se ha realizado un ciclo de aprendizaje

con tasa 0,01 y un segundo ciclo con una tasa de 0,05. Después, se han descongelado las dos últimas capas y se ha entrenado un ciclo donde la primera capa tiene una tasa de aprendizaje de 0,001 y la última una tasa de 0,01 (la tasa de las capas intermedias se calcula geométricamente). Seguidamente, se han descongelado otra capa más y se ha realizado un ciclo de aprendizaje con tasas de 0,0005 a 0,005. Con el modelo entero descongelado se han llevado a cabo dos ciclos de aprendizaje con una tasa de 0,0001 en la primera capa y 0,001 en la última. Por último, se han realizado dos ciclos con una tasa de 0,001 en toda la red.

El resultado final de 63,67% es muy bueno, superior a todos los anteriores, y se obtiene una matriz de confusión muy similar a la del resto de modelos.

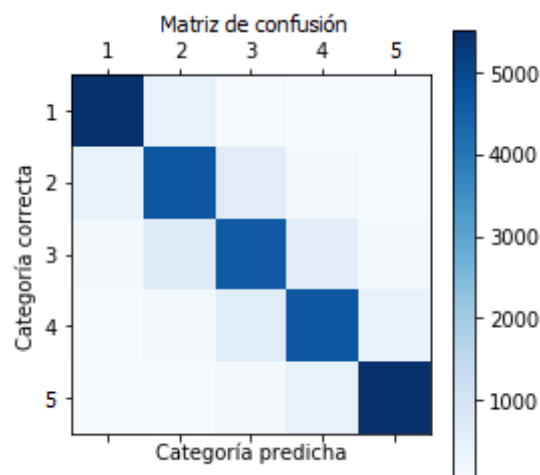


Fig. 5.14. Matriz de confusión de ULMFiT en el conjunto multiclase

5.5. Discusión y comparativa final de modelos

Vistos los resultados de la experimentación con todos los modelos, se procede a realizar una valoración comparativa de los modelos finales.

A continuación, se puede visualizar el resumen de los mejores resultados obtenidos para cada uno de los modelos junto a una gráfica comparativa de los mismos.

TABLA 5.19. RESULTADOS FINALES EN EL CONJUNTO BINARIO

	Accuracy	F1	ROC-AUC	Loss
SVM (bag-of-words)	87,22	87,19	0,9452	0,3008
DPCNN (word embed.)	87,35	87,06	0,9477	0,3141
BLSTM (word embed.)	88,00	88,00	0,9514	0,2924
ULMFiT	89,72	89,52	0,9647	0,2680

TABLA 5.20. RESULTADOS FINALES EN EL CONJUNTO MULTICLASE

	Accuracy	F1	ROC-AUC	Loss
SVM (bag-of-words)	59,65	59,25	0,8652	0,9658
DPCNN (word embed.)	61,03	60,49	0,8811	0,9368
BLSTM (word embed.)	61,68	60,25	0,8704	0,9025
ULMFiT	63,67	62,98	0,8859	0,8753

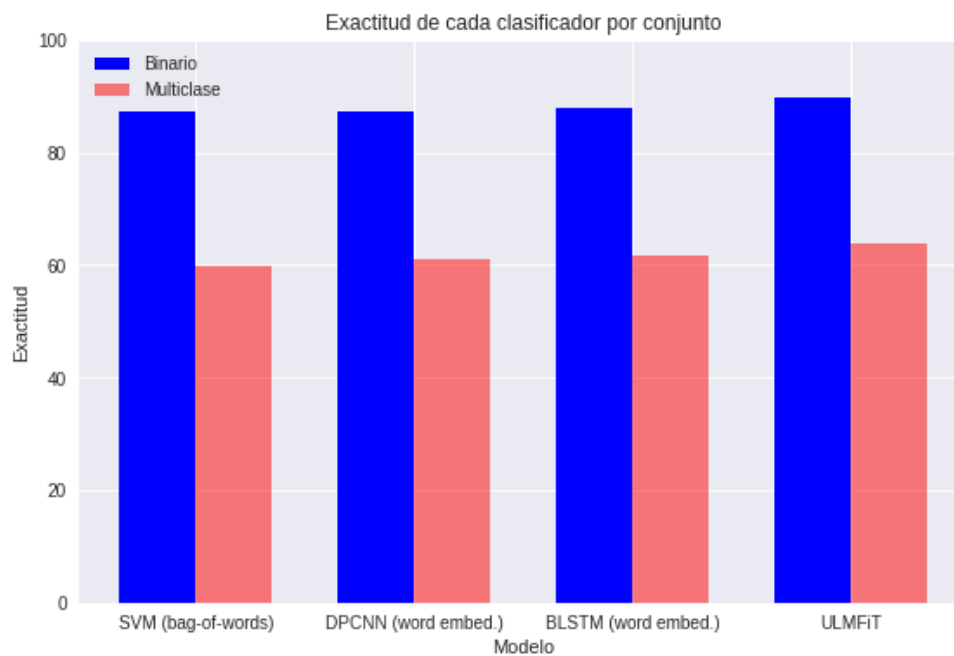


Fig. 5.15. Comparativa de modelos

El mejor modelo resultante ha sido el obtenido con ULMFiT en ambos conjuntos, el binario y el multiclase. En el caso del binario, se ha conseguido un modelo que logra clasificar la polaridad de una reseña con una exactitud de un 89,72 %, lo cuál es un muy buen resultado, pues es cercano a los resultados del estado del arte, y que mejora en un 2,5 % el mejor modelo obtenido con el modelo *bag-of-words* y en un 1,72 % al mejor modelo obtenido utilizando *word embeddings* ya entrenados.

Por el lado del conjunto multiclase, se observa la misma tesitura, aunque en este caso ULMFiT ha funcionado mejor (o los otros modelos peor), pues la diferencia entre el modelo obtenido con máquinas de soporte vectorial y ULMFiT ha sido de un 4,02 %. Los modelos obtenidos arrojan una conclusión muy positiva, pues logran clasificar las reseñas en 5 estrellas con hasta un 63,67 % de exactitud y, aunque pudiera parecer que es un porcentaje lejano a los obtenidos con el conjunto binario, conseguir un modelo que clasifique correctamente 5 categorías distintas de texto es complejo, lo cuál se ha visto en la sección 2.5, pues los modelos del estado del arte siguen tratando de resolver esta situación, alcanzando ULMFiT el mejor resultado con un 70 %.

Particularmente entre las redes de *word embeddings*, ha obtenido mejor resultado la que emplea LSTM, mismo tipo de red que utiliza el mejor modelo del estado del arte (ULMFiT), aunque a priori se esperaba que el modelo DPCNN planteado por Johnson y Zhang (2017) tuviera mejores resultados de acuerdo a los que mencionaban los autores en su publicación.

Ésta valoración se puede ver en mejor perspectiva si comparamos los resultados obtenidos con aquellos que han obtenido los autores de los modelos que se ha tratado de replicar (vistos en la tabla 2.2).

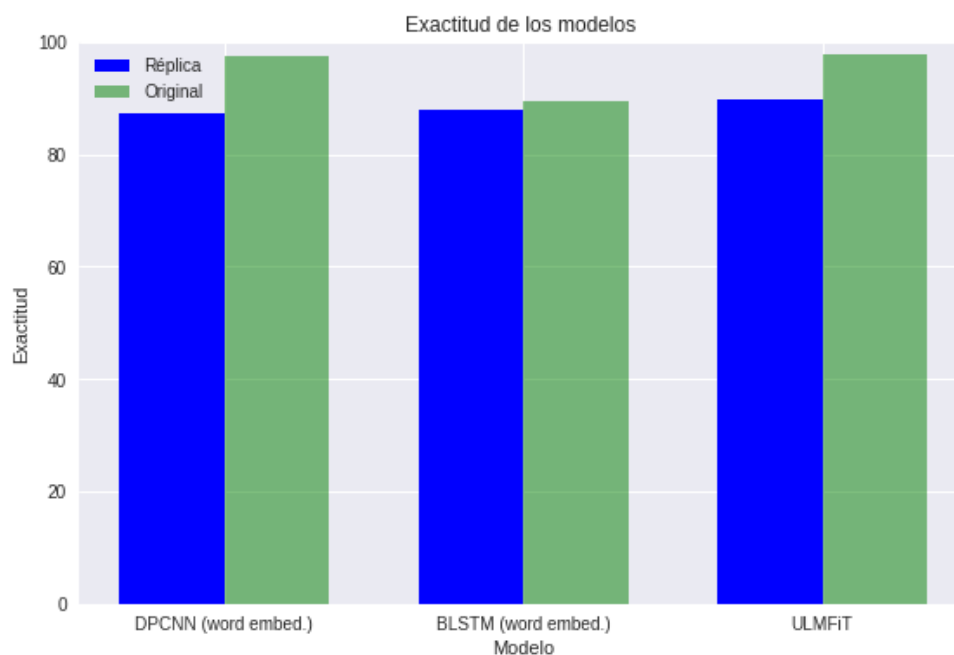


Fig. 5.16. Comparativa de modelos con resultados originales

Teniendo esto en cuenta, aunque DPCNN obtenía muy buenos resultados, cercanos a los de ULMFiT, no se ha conseguido un modelo cercano a esos resultados, habiendo una diferencia de un 10,01 % de exactitud respecto a nuestro modelo. En cambio, para el modelo BLSTM sólo existe una diferencia del 1,5 %, habiendo logrado un resultado similar, si bien es cierto que los resultados que habían obtenido los autores de este modelo eran bastante inferiores a los de DPCNN.

Como se esperaba, ULMFiT, que era el modelo con mejores resultados del estado del arte, ha sido también el modelo que mejor solución ha dado a nuestro problema, aunque también ha existido cierta diferencia respecto a los resultados que planteaban los autores del estudio, un 8,12 %.

En general, y evaluando los modelos más allá de los porcentajes de exactitud, se han obtenido modelos definitivos con muy buenos resultados. Antes de la experimentación, podría pensarse que los algoritmos de *machine learning* tradicionales con el modelo *bag-of-words* iban a ofrecer peor rendimiento, pero han resultado en modelos muy buenos y que se acercan bastante a los que emplean *word embeddings*, aunque para estos últimos se han obtenido peores resultados que los de sus creadores, los cuáles si que mejoran en mayor grado a los modelos *bag-of-words*.

En el contexto de solucionar el problema planteado en este trabajo de clasificar las reseñas, se emplearía para su explotación el mejor modelo obtenido con ULMFiT. No obstante, se debe valorar que los algoritmos con el modelo *bag-of-words*, habiendo obtenido buenos resultados, son modelos cuya complejidad de entrenamiento es muy inferior a las redes neuronales empleadas para los *word embeddings* y su tiempo de entrenamiento también.

6. MARCO REGULADOR

En este capítulo se realiza un análisis de las regulaciones aplicables al proyecto realizado.

6.1. Legislación aplicable

A la hora de desarrollar un proyecto, es requisito indispensable realizar una análisis de la normativa existente que regula el campo en el que se enmarca dicho proyecto.

En el caso de nuestro estudio, el campo de la inteligencia artificial, y más concretamente, del aprendizaje automático, no existen normativas desarrolladas. Más aún, al tratarse de un trabajo de investigación, y no de un desarrollo de un componente que se pudiera industrializar, el producto final del mismo no es más que un resultado, no un producto, por lo que no tendría que acogerse a la regulación de un sector comercial.

En el plano de la inteligencia artificial, hasta ahora no ha existido ningún tipo de regulación a nivel europeo que estableciera pautas sobre este campo. Recientemente, a finales del año 2018, la Comisión Europea, concretamente el Mercado Único Digital, sector de la Comisión que abarca servicios digitales en línea, respondiendo al crecimiento de este campo, ha presentado (European Commission, 2018) un plan de actuación para incentivar el desarrollo del área, pero al mismo tiempo implantar una legislación y una serie de directrices éticas.

Como se puede ver en la figura 5.1, en los inicios de 2019 se plantea la discusión de un boceto de las directrices que se quieren implantar. En este sentido, actualmente no aplica ninguna restricción legal sobre este campo.

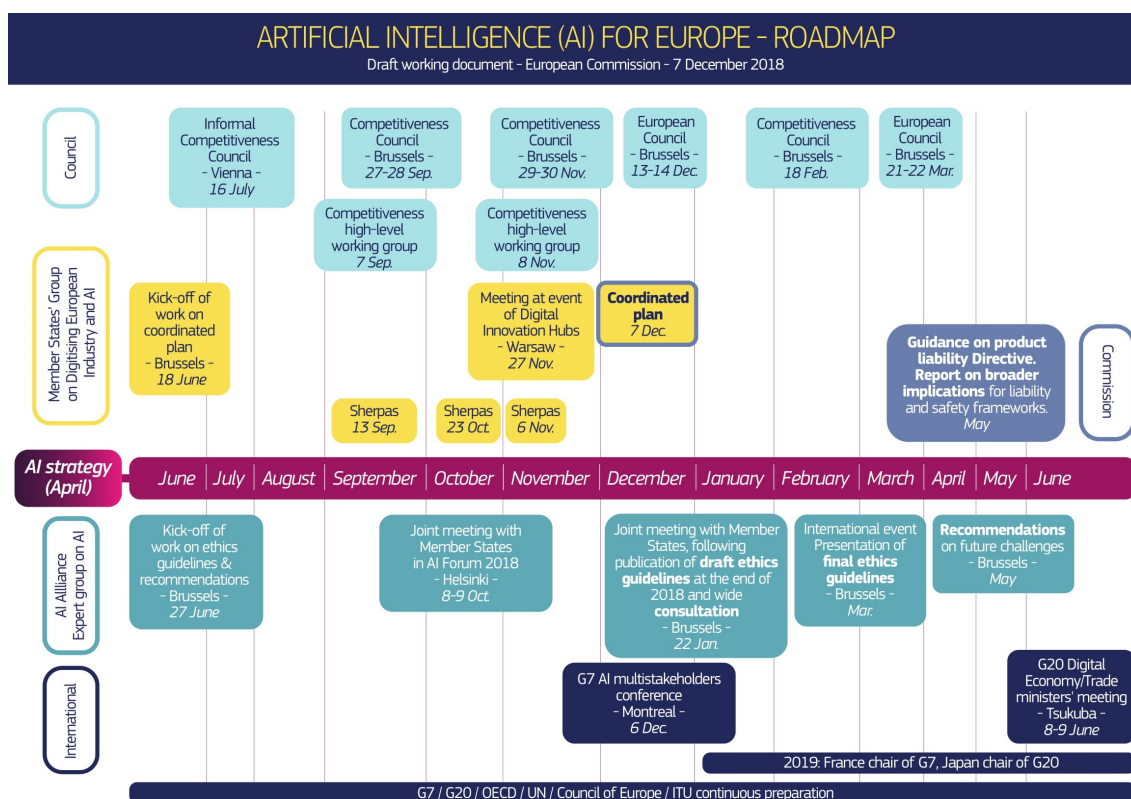


Fig. 6.1. Planificación de la gestión de la IA en Europa

Por otro lado, se es consciente de que en este trabajo se usan diversos datos, por lo que es necesario revisar la regulación aplicable al tratamiento de datos. En Europa, el tratamiento de los datos se regula mediante el “Reglamento General de Protección de Datos” (RGPD), el cuál en España se ha adaptado en la “Ley Orgánica de Protección de Datos Personales y garantías de los derechos digitales” (Agencia Estatal Boletín Oficial del Estado, 2018). Esta ley regula la protección de las personas físicas respecto al tratamiento de sus datos personales y, adicionalmente, estipula una serie de derechos que tienen los usuarios en el uso de contenido digital.

En el contexto de ese estudio, el conjunto de datos de Yelp contiene datos sobre los usuarios de la plataforma. No obstante, estos datos no son de carácter personal, sino relativos al uso del sitio web, por lo que no sería necesaria la aplicación del reglamento. Además, durante este trabajo se han descartado los datos de los usuarios por no ser útiles, empleando únicamente los datos de las reseñas.

Respecto al propio conjunto, Yelp describe unos términos y condiciones de uso que regulan la responsabilidad que se tiene a la hora de emplear su conjunto de datos.

En base a estos términos y condiciones, existen una serie de restricciones a las cuáles se ha acogido este proyecto:

- No se redistribuirán o mostrarán los datos a terceras partes, ni se utilizarán para crear un negocio en base a la información existente.
- No se utilizarán los datos de ninguna forma comercial.
- No se utilizarán los datos de manera que violen alguna regulación o derecho de alguna persona, incluyendo derechos de propiedad intelectual o de privacidad, o de alguna manera que pueda ser perjudicial para Yelp, sus proveedores o sus usuarios finales.
- No se redistribuirá o revelarán resúmenes, análisis o métricas acerca de los datos a terceras partes, salvo con fines académicos o de investigación.
- No se utilizarán los datos con naturaleza competitiva contra Yelp.
- No se mostrarán los datos de una manera que pudiera insinuar cualquier tipo de afiliación o patrocinio con Yelp.
- No se venderá, prestará, alquilará o sublicenciará ninguna parte de los datos.
- No se publicará, de ninguna manera, parte de los datos de forma que se menosprecie a Yelp o a alguno de sus productos, o que se infrinja su propiedad intelectual.
- No se utilizarán los datos de manera que se pueda interpretar que Yelp es responsable de cualquier creación a partir de los mismos, o que se pueda interpretar que se corresponde con las opiniones de Yelp.
- No se utilizarán los datos para cualquier propósito prohibido por la ley.

6.2. Estándares técnicos y propiedad intelectual

En esta sección se estudian los estándares técnicos y las cuestiones relativas a la propiedad intelectual del trabajo realizado y de las herramientas utilizadas.

Dentro del campo del aprendizaje automático, o de la inteligencia artificial en su conjunto, existen una serie de procesos estandarizados a la hora de desarrollar un modelo por parte de los investigadores. No obstante, estos procesos están estandarizados por su uso común, no porque exista una organización internacional de estandarización que haya creado una normal general, por lo que ocurre lo mismo que en el caso de la legislación.

Sin embargo, en el año 2017, el comité ISO JTC 1 de la ISO (International Organization for Standardization), centrado en el desarrollo de estándares dentro del campo de las tecnologías de la información, planteó el desarrollo la norma ISO/IEC JTC 1/SC 42 cuyo objetivo es estandarizar el campo de la inteligencia artificial, construyendo un programa que sirva como guía para el proceso de desarrollo de aplicaciones de inteligencia artificial. Por tanto, en el momento de la redacción de este trabajo la norma se encuentra en desarrollo, esperando su finalización en 2021 (Diab, 2018).

Por otra parte, es necesario describir lo relativo a la propiedad intelectual.

Este trabajo, en el marco de un trabajo académico, se encuentra sujeto a la licencia Creative Commons de Reconocimiento - No Comercial - Sin Obra Derivada. Esto significa que se puede hacer uso (copiando, distribuyendo o representando) del mismo siempre que se cite y reconozca al autor, que no se haga con fines comerciales, y que no se produzcan obras derivadas del mismo. Más allá de esta licencia, el estudio no contiene ninguna idea patentada cuyo derecho se deba indicar.

Sobre las herramientas utilizadas, descritas en la sección 3.4, se van a describir los permisos de uso sobre las mismas. En la tabla 5.1 se puede ver un resumen.

Todas las versiones del lenguaje Python son de código libre y las más recientes (desde 2001) cuentan con la licencia GNU Public License.

Lo mismo ocurre con todas sus librerías utilizadas. En el caso de Scikit-learn, ésta cuenta con una licencia BSD de 3 cláusulas, la cuál es una licencia libre que permite el uso total y la redistribución siempre que se cumplan tres cláusulas: las redistribuciones del código deben contar con el aviso de copyright y las condiciones de uso, las redistribuciones del módulo deben contar con el aviso de copyright y las condiciones en su documentación, y el nombre de la herramienta no se usará para promocionar

ningún producto. La librería Pandas también utiliza esta licencia.

Por su parte, el uso de la librería de procesamiento de lenguaje NLTK y de fastai, están sujetos a la licencia Apache 2.0, que permite el uso libre pero estipula que en las redistribuciones de código se debe mantener la licencia. Keras, la librería de aprendizaje profundo, está sujeta a otra licencia libre, la licencia MIT, muy similar a la BSD.

La tecnología que se ha utilizado para escribir la memoria, \LaTeX , tiene su propia licencia, llamada Licencia Pública del Proyecto LaTeX (LPPL) y es de uso libre.

TABLA 6.1. RESUMEN DE LICENCIAS DE USO DE LAS
TECNOLOGÍAS EMPLEADAS

Tecnología	Licencia	Tipo
Python	GPL	Libre
Scikit-learn	BSD	Libre
Pandas	BSD	Libre
NLTK	Apache 2	Libre
fastai	Apache 2	Libre
Keras	MIT	Libre
\LaTeX	LPPL	Libre

Como se puede observar, todas las tecnologías utilizadas son de uso libre, por lo que no existe ningún problema de propiedad intelectual.

7. ENTORNO SOCIO-ECONÓMICO

A continuación, se incluye el detalle de la planificación de las actividades que conforman el proyecto, junto a los costes de elaboración.

Después, se realiza un análisis de las aplicaciones prácticas que podrían tener los resultados obtenidos y el posible impacto de las mismas.

7.1. Planificación del proyecto

En esta sección se expone la planificación que se ha llevado a cabo para este trabajo. En un primer lugar, se describirán todas las actividades planteadas para alcanzar el objetivo final y, después, se representará la gestión que se ha hecho.

Planifica la ejecución temporal de las actividades que componen el proyecto es necesario para tener un control sobre el estado de las mismas, pudiendo conocer si están retrasadas o adelantadas y así poder realizar los ajustes necesarios para llegar a tiempo a la fecha tope.

Existen varios hitos importantes marcados en la planificación. La primera toma de contacto para plantear el objeto de estudio se tuvo el jueves 8 de febrero de 2018, en la cuál se disertó sobre las distintas posibilidades que había. No obstante, el contexto del problema no se decidió hasta la siguiente reunión, el 23 de febrero de 2018, en la cuál se trató la existencia del conjunto de datos de Yelp y los posibles enfoques que ofrecía, los cuáles eran numerosos. El tiempo siguiente a esta reunión se dedicó a pensar sobre qué tipo de estudio realizar sobre el conjunto de datos y la falta de tiempo provocó que el inicio real del proyecto, con la idea definida, no se produjera hasta el 5 de septiembre de 2018, hito de inicio. Teniendo en cuenta que la fecha de entrega viene dada por la Universidad, el hito final es el 21 de febrero de 2019.

Las actividades realizadas vienen dadas, en mayor medida, por la metodología de desarrollo planteada (ver figura 3.1) y los capítulos de la memoria, que representan

el desglose del proyecto en apartados. Se pueden ver descritas a continuación:

- Gestión del proyecto: es la tarea que se mantiene durante todo el proyecto, desde el hito de inicio, que corresponde a la primera reunión dónde se marcó finalmente el hilo del estudio.
- Preparación del proyecto: comprende las tareas que tratan todo el análisis previo al comienzo de la realización del proyecto.
 - Análisis del problema
 - Revisión de la literatura
 - Planteamiento del proceso
- Escritura de la memoria: actividad paralela a la realización del proyecto que consiste en documentar todo el trabajo que va haciendo.
- Preparación de los datos: fase que abarca todo el tratamiento de los datos antes de comenzar la experimentación con los mismos.
 - Obtención de los datos
 - Análisis de los datos
 - Procesamiento de los datos
- Experimentación: desarrollo principal del proyecto, la experimentación con los modelos planteados.
 - Planteamiento de la experimentación
 - Implementación de los modelos
 - Realización de la experimentación
- Evaluación de modelos: análisis de los resultados obtenidos y conclusiones del estudio en base a los mismos.
 - Comparación de resultados

- Conclusiones

- Fecha de entrega: hito final del proyecto.

Siendo estas tareas las que conforman el proyecto, se ha diseñado una planificación de la realización de las mismas para controlar el avance del trabajo. El resultado de la planificación es el diagrama de Gantt de la figura 7.1.

Como se puede observar, el proyecto está comprendido entre el 5 de septiembre de 2018 y el 21 de febrero de 2019, día de la entrega, lo que correspondería a 170 días. No obstante, el proyecto se concluye el 15 de febrero y en la fase de “Experimentación” existe un receso de 6 días, por lo que el total de días reales que se ha trabajado en el proyecto ha sido de **158 días**. Habiendo empleado una media de 3 horas/día a su realización, el tiempo total de elaboración del trabajo es de **474 horas**.

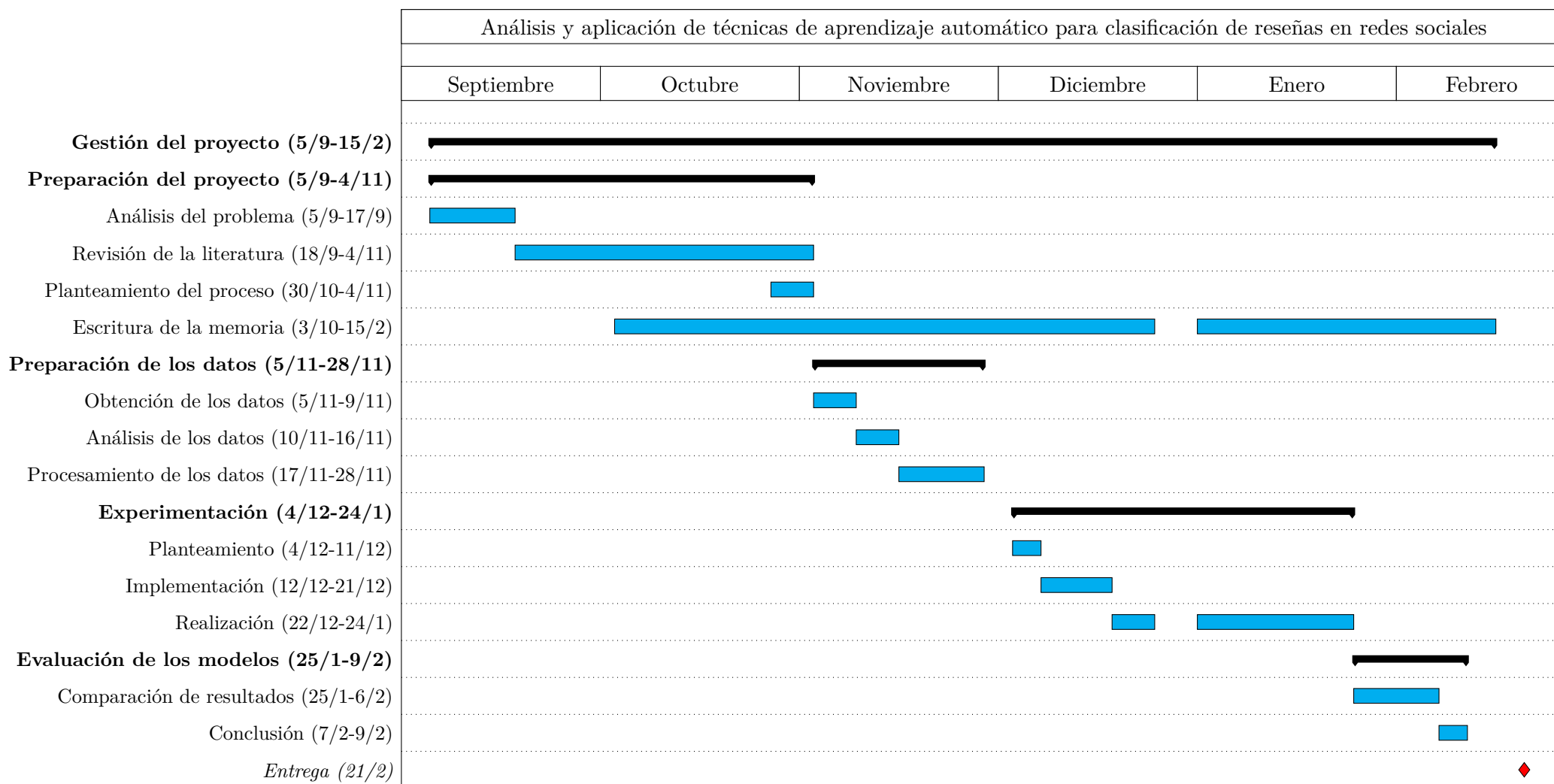


Fig. 7.1. Diagrama de Gantt de planificación

7.2. Presupuesto del proyecto

En este apartado se incluye una estimación del presupuesto del proyecto en base a las actividades marcadas en la planificación anterior que permite conocer los costes de la realización del mismo.

7.2.1. Coste de personal

El coste de personal corresponde al coste del esfuerzo humano realizado para llevar a cabo el proyecto, es decir, el salario de los recursos humanos.

Para el cálculo del salario correspondiente se estima un salario de 30 euros/hora y se tienen en cuenta las horas de acuerdo a la planificación de la sección anterior. El resultado del coste de personal se puede ver en la siguiente tabla:

TABLA 7.1. COSTE DE PERSONAL

Rol	Empleados	Horas	Salario/hora	Coste (€)
Investigador	1	474	30	14.220
TOTAL	1	474	30	14.220

7.2.2. Coste de material

El coste de material corresponde al coste de todos los recursos materiales empleados para la realización del proyecto.

El coste de las herramientas de software utilizadas es nulo, debido a que todas son de uso libre y gratuito. Por otro lado, se ha empleado un ordenador portátil MSI adquirido por 1.050 euros, por lo que es necesario conocer su coste imputable. Dicho coste se calcula de la siguiente forma:

$$\text{Coste imputable} = \left(\frac{\text{período de uso}}{\text{período útil}} \right) * \% \text{ de uso} * \text{coste} \quad (7.1)$$

Se estima que el ordenador portátil tenga una vida útil de 6 años, por lo que el coste

material sería el siguiente:

TABLA 7.2. COSTE DE MATERIAL

Material	Coste	Tiempo de uso	Tiempo útil	% de uso	Coste imputable
Portátil MSI	1.050 €	5,7 meses	72 meses	100	83,125 €
TOTAL	1.050 €	5,7 meses	72 meses	100	83,125 €

7.2.3. Costes adicionales

En este cálculo se han tenido en cuenta otros costes añadidos a los anteriores:

- Material fungible: bolis y papeles que se han empleado para la realización del proyecto.
- Viajes: los viajes a la universidad para las reuniones con el tutor, los cuáles se han hecho en vehículo propio. La distancia a la universidad es de 7 km, 14 en viaje de ida y vuelta, se estima un coste de 10 céntimos por kilómetro y se han tenido 8 reuniones.

Concepto	Coste
Material fungible	5 €
Viajes	11,2 €
TOTAL	16,2 €

TABLA 7.3. COSTES ADICIONALES

7.2.4. Coste final

A continuación, se muestra el resultado del cálculo del presupuesto para la realización del proyecto. Dicho resultado es la acumulación de los costes calculados. Adicionalmente, se han calculado unos **costes indirectos** para cubrir todas las necesidades del proceso ajenas a la investigación (teléfono, internet, energía) y cualquier gasto inesperado no contemplado. A estos costes se les concede un 10 % de los costes directos.

TABLA 7.4. COSTE FINAL

Concepto	Coste bruto	Coste total
Coste de personal	14.220 €	14.220 €
Coste de material	83,13 €	83,13 €
Costes adicionales	16,2 €	16,2 €
Costes indirectos	957,93 €	957,93 €
TOTAL BRUTO	15.277,26 €	15.277,26 €
Margen de riesgo (25 %)	3.819,32 €	19.096,58 €
Margen de beneficio (15 %)	2.864,49 €	21.961,07 €
IVA (21 %)	4.611,82 €	26.572,89 €
TOTAL		26.572,89 €

Por tanto, se concluye que la realización del proyecto tendría un coste estimado de 26.572,89 euros.

7.3. Impacto

Es necesario plantear las posibles aplicaciones prácticas que tiene este estudio y el impacto socio-económico que podría generar en el sector.

Principalmente, se ha tratado de obtener un modelo de análisis de sentimiento que obtuviese resultados cercanos a los del estado del arte, por lo que se han intentado replica algunos de los modelos con mejores resultados. De esta forma, el modelo contribuye al sector del aprendizaje automático para análisis de sentimiento como una comparativa práctica de los distintos tipos de modelos que existen.

Por otra parte, la utilización del modelo resultante ofrece diversas posibilidades que pueden ser muy beneficiosas. El modelo puede ser empleado por los propietarios de los negocios para analizar las reseñas sobre su negocio y poder detectar virtudes y defectos, ofreciendo un mejor servicio a los clientes y aumentando la calidad del producto. Asimismo, proporciona a las corporaciones interesadas la capacidad de incorporarlo a sus análisis de *business intelligence* para obtener datos acerca de los

gustos y opiniones de los usuarios, sin la necesidad de analizar cada reseña/opinión de forma individual.

En relación a lo mencionado, el modelo se puede emplear como parte de estudios de mercado, permitiendo obtener una perspectiva de las necesidades de los consumidores y de formas de satisfacerlas. En este caso, si el estudio de mercado conduce a una campaña de marketing posterior, una vez realizada se puede también utilizar el modelo de análisis de sentimiento para medir el retorno de la campaña.

Otro de los sectores en los que tendría impacto sería en el servicio de soporte al consumidor, donde se podría comprender de mejor manera los problemas de los consumidores.

En el contexto de las aplicaciones o sitios de reseñas y opinión, el uso del modelo podría ser muy útil, pues podría ayudar a la detección de comentarios muy negativos o que intenten perjudicar a los servicios valorados, mejorando la calidad de las reseñas, y también podría emplearse para conseguir una mayor fiabilidad de las reseñas, tratando de detectar reseñas cuyo texto no coincide con la estrella correspondiente y alertando a los usuarios para que lo corrijan si se trata de un error.

Así, aplicando el modelo en estas opciones, se conseguiría abaratar costes, agilizando los procesos y creando valor para los que lo utilicen.

8. CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

En este trabajo se ha resuelto un problema de clasificación de reseñas provenientes de una red social mediante la utilización de distintos tipos de modelos de aprendizaje automático. Se han estudiado y aplicado algoritmos que emplean el modelo *bag-of-words*, redes neuronales que utilizan *word embeddings*, y un modelo, ULM-FiT, encuadrado dentro de los nuevos modelos que tratan de universalizar los *word embeddings* entrenados para su utilización en múltiples tareas.

A través de ellos, se han logrado distintos modelos capaces de clasificar las reseñas tanto en positivas y negativas como en 5 estrellas con resultados cercanos a los del estado del arte, con un 89,72 % de exactitud para el primer caso y un 63,67 % para el segundo. De esta forma, no solo se ha resuelto el problema planteado sino que se logra observar y comparar los distintos resultados que ofrece cada tipo de modelo.

Además de la resolución de la cuestión, se ha logrado profundizar los conocimientos sobre los distintos modelos de aprendizaje automático y, más particularmente, de análisis de sentimiento, complementando así aquellos obtenidos durante la realización del grado, y también se ha descrito de forma detallada todo el proceso llevado a cabo para la obtención de los modelos, de manera que todos son replicables, abriendo así la posibilidad a la ampliación de los experimentos.

Considerando el trabajo en su conjunto, se ha llevado a cabo un trabajo de investigación, estudiando en primer lugar la literatura relativa al problema que se quiere resolver, analizando la misma para aplicarla al problema y realizando una experimentación de acuerdo a los métodos y técnicas aprendidas durante el grado. Siendo éste el núcleo principal del trabajo, se han realizado también tareas inherentes a este tipo de proyecto, analizando las regulaciones aplicables y el impacto socio económico, así como elaborando un presupuesto.

Respecto a las líneas de trabajo que podrían surgir en el futuro a partir de lo planteado en este documento, en primer lugar sería posible la aplicación de los modelos utilizados en otros conjuntos de datos de otros sitios o redes sociales de

reseñas, como por ejemplo TripAdvisor, muy similar al conjunto utilizado de Yelp, o de IMDB o Rotten Tomatoes, pasando a reseñas de otro tipo, en este caso películas, tanto para ver cómo se comporta el modelo en otro contexto diferente al entrenado como para su aplicación en esos otros sitios.

Del mismo modo, podría llevarse a cabo la explotación del modelo definitivo, utilizándolo para las múltiples posibles aplicaciones mencionadas en el documento, y también sería muy interesante probar el resto de modelos recientes que existen en el estado del arte (sección 2.4), como ELMo, Universal Sentence Encoder o BERT, e incluso utilizar otro tipo de word embeddings, como GloVe o InferSent.

Teniendo en cuenta que existen muchas maneras de experimentar y entrenar una red neuronal, y que, como se ha mencionado, los modelos obtenidos son totalmente replicables, podría tratarse de ahondar en la experimentación probando parámetros no utilizados e incluso probar nuevos tipos de modelos y arquitecturas, dado que en este trabajo solo se han cubierto dos y existen muchas más arquitecturas que han dado buenos resultados, como se ha visto en el capítulo del estado del arte. Así, podría tenerse una mejor perspectiva sobre el funcionamiento que ofrecen las distintas arquitecturas y, posiblemente, encontrar un modelo que funcione todavía mejor.

También podrían plantearse los mismos modelos pero centrando mucho más la atención en la preparación de los datos, realizando tareas de limpieza del texto más minuciosas o tratando de probar nuevos preprocesamientos que den lugar a un mejor texto de entrada, lo cuál podría ofrecer mejores resultados. Por supuesto, como el conjunto de datos utilizado se ha reducido en esta etapa para que sea más rápido de realizar el modelado, existiría también la posibilidad de utilizar conjuntos con muchas más reseñas que mejoren en cierta manera los porcentajes de exactitud.

Fuera del análisis de sentimiento, pero continuando con el objetivo de obtener información a partir del conjunto de datos de Yelp, éste contiene información acerca de los servicios y características que ofrece cada negocio, por lo que podría tratarse de estudiar de qué forma éstos afectan a la valoración de los usuarios y entrenar modelos que traten de reconocer estos patrones.

BIBLIOGRAFÍA

- Agencia Estatal Boletín Oficial del Estado. (2018). Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales. En *BOE núm. 294*.
- Altszyler, E., Sigman, M. & Slezak, D. F. (2017). The interpretation of dream meaning: Resolving ambiguity using Latent Semantic Analysis in a small corpus of text. *Consciousness and Cognition*, 56, 178-187.
- Bird, S., Klein, E. & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc.
- Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *CoRR*, *abs/1607.04606*. Recuperado desde <http://arxiv.org/abs/1607.04606>
- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32. Recuperado desde <https://doi.org/10.1023/A:1010933404324>
- Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R. S., ... Kurzweil, R. (2018). Universal Sentence Encoder. *CoRR*, *abs/1803.11175*. Recuperado desde <http://arxiv.org/abs/1803.11175>
- Chollet, F. et al. (2015). Keras. <https://keras.io>.
- Conneau, A., Kiela, D., Schwenk, H., Barrault, L. & Bordes, A. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *CoRR*, *abs/1705.02364*. Recuperado desde <http://arxiv.org/abs/1705.02364>
- Dave, K., Lawrence, S. & Pennock, D. M. (2003). Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. En *Proceedings of the 12th international conference on World Wide Web* (pp. 519-528).
- Devlin, J., Chang, M., Lee, K. & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, *abs/1810.04805*. Recuperado desde <http://arxiv.org/abs/1810.04805>
- Diab, W. (2018). About JTC 1/SC 42 Artificial intelligence. Recuperado desde <https://jtc1info.org/jtc1-press-committee-info-about-jtc-1-sc-42/>

- Domingos, P. (2012). A Few Useful Things to Know About Machine Learning. *Commun. ACM*, 78-87. Recuperado desde <http://doi.acm.org/10.1145/2347736.2347755>
- European Comission. (2018). Member States and Commission to work together to boost artificial intelligence “made in Europe”. Recuperado desde http://europa.eu/rapid/press-release_IP-18-6689_en.htm
- Goldberg, Y. (2015). A Primer on Neural Network Models for Natural Language Processing. *CoRR*, *abs/1510.00726*. Recuperado desde <http://arxiv.org/abs/1510.00726>
- Guzman, E. & Maalej, W. (2014). How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. En *2014 IEEE 22nd International Requirements Engineering Conference (RE)* (pp. 153-162).
- Harris, Z. S. (1954). Distributional Structure. *WORD*, 10, 146-162. Recuperado desde <https://doi.org/10.1080/00437956.1954.11659520>
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, *abs/1207.0580*. Recuperado desde <http://arxiv.org/abs/1207.0580>
- Ho, T. K. (2002). A Data Complexity Analysis of Comparative Advantages of Decision Forest Constructors. *Pattern Analysis and Applications*, 5, 102-112.
- Howard, J. & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *CoRR*, *abs/1801.06146*. Recuperado desde <http://arxiv.org/abs/1801.06146>
- Ifrim, G., Bakir, G. & Weikum, G. (2008). Fast logistic regression for text categorization with variable-length n-grams. *Bing Liu, Bing; Sarawagi, Sunita; Li, Ying: KDD 2008 : proceedings of the 14th ACM KDD International Conference on Knowledge Discovery and Data Mining, ACM*, 354-362.
- Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. En *Proceedings of the 10th European Conference on Machine Learning* (pp. 137-142). Recuperado desde <https://doi.org/10.1007/BFb0026683>
- Johnson, R. & Zhang, T. (2014). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. *CoRR*, *abs/1412.1058*. Recuperado desde <http://arxiv.org/abs/1412.1058>

- Johnson, R. & Zhang, T. (2016). Supervised and Semi-supervised Text Categorization Using LSTM for Region Embeddings. En *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (pp. 526-534). Recuperado desde <https://arxiv.org/pdf/1602.02373.pdf>
- Johnson, R. & Zhang, T. (2017). Deep Pyramid Convolutional Neural Networks for Text Categorization. En *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 562-570). Recuperado desde <http://aclweb.org/anthology/P17-1052>
- Kemp, S. (2018). Digital in 2018: World's Internet users pass the 4 billion mark. Recuperado desde <https://wearesocial.com/blog/2018/01/global-digital-report-2018>
- Kolchyna, O., Souza, T. T. P., Treleaven, P. C. & Aste, T. (2015). Twitter Sentiment Analysis: Lexicon Method, Machine Learning Method and Their Combination. *CoRR*, *abs/1507.00955*. Recuperado desde <http://arxiv.org/abs/1507.00955>
- Leskovec, J., Rajaraman, A. & Ullman, J. D. (2014). *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press.
- Li, S. (2018). Web Scraping TripAdvisor, Text Mining and Sentiment Analysis for Hotel Reviews. Recuperado desde <https://towardsdatascience.com/scraping-tripadvisor-text-mining-and-sentiment-analysis-for-hotel-reviews-cc4e20aef333>
- Lin, C. & He, Y. (2009). Joint Sentiment/Topic Model for Sentiment Analysis. En *Proceedings of the 18th ACM Conference on Information and Knowledge Management* (pp. 375-384). Recuperado desde <http://doi.acm.org/10.1145/1645953.1646003>
- Liu, B. (2010). Sentiment analysis and subjectivity. En *Handbook of Natural Language Processing, Second Edition*. Taylor and Francis Group, Boca.
- Loh, W.-Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*(1), 14-23. Recuperado desde <https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.8>
- Maclin, R. & Opitz, D. W. (2011). Popular Ensemble Methods: An Empirical Study. Recuperado desde <http://arxiv.org/abs/1106.0257>

- Mäntylä, M. V., Graziotin, D. & Kuuttila, M. (2016). The Evolution of Sentiment Analysis - A Review of Research Topics, Venues, and Top Cited Papers. *CoRR*, *abs/1612.01556*. Recuperado desde <http://arxiv.org/abs/1612.01556>
- Mccallum, A. & Nigam, K. (2001). A Comparison of Event Models for Naive Bayes Text Classification. Recuperado desde <http://www.kamalnigam.com/papers/multinomial-aaaiws98.pdf>
- McCann, B., Bradbury, J., Xiong, C. & Socher, R. (2017). Learned in Translation: Contextualized Word Vectors. *CoRR*, *abs/1708.00107*. Recuperado desde <http://arxiv.org/abs/1708.00107>
- McKinney, W. (2010). Data Structures for Statistical Computing in Python. En S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 51-56).
- Merity, S., Keskar, N. S. & Socher, R. (2017). Regularizing and Optimizing LSTM Language Models. *CoRR*, *abs/1708.02182*. Recuperado desde <http://arxiv.org/abs/1708.02182>
- Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *CoRR*, *abs/1301.3781*. Recuperado desde <http://arxiv.org/abs/1301.3781>
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, *abs/1310.4546*. Recuperado desde <http://arxiv.org/abs/1310.4546>
- Mount, J. (2012). The equivalence of logistic regression and maximum entropy models. Recuperado desde <http://www.win-vector.com/dfiles/LogisticRegressionMaxEnt.pdf>
- Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F. & Stoyanov, V. (2016). SemEval-2016 task 4: Sentiment analysis in Twitter. En *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)* (pp. 1-18).
- Ng, A. (2017). CS229 Lecture notes.
- Pang, B. & Lee, L. (2004). A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts. *Computing Research Repository - CORR*, 271-278. Recuperado desde <http://www.cs.cornell.edu/home/llee/papers/cutsent.pdf>

- Pang, B. & Lee, L. (2008). Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval*, 2(1-2), 1-135. Recuperado desde <http://www.cs.cornell.edu/home/llee/omsa/omsa.pdf>
- Pang, B., Lee, L. & Vaithyanathan, S. (2002). Thumbs Up?: Sentiment Classification Using Machine Learning Techniques. En *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing* (pp. 79-86). Association for Computational Linguistics. Recuperado desde <https://doi.org/10.3115/1118693.1118704>
- Parikh, R. & Movassate, M. (2009). Sentiment Analysis of User-Generated Twitter Updates using Various Classification Techniques. Recuperado desde <https://nlp.stanford.edu/courses/cs224n/2009/fp/19.pdf>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
- Pennington, J., Socher, R. & Manning, C. (2014). Glove: Global Vectors for Word Representation. (Vol. 14, pp. 1532-1543). Recuperado desde <https://nlp.stanford.edu/pubs/glove.pdf>
- Perone, C. S., Silveira, R. & Paula, T. S. (2018). Evaluation of sentence embeddings in downstream and linguistic probing tasks. *CoRR*, abs/1806.06259. Recuperado desde <http://arxiv.org/abs/1806.06259>
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. & Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, abs/1802.05365. Recuperado desde <http://arxiv.org/abs/1802.05365>
- Raschka, S. (2014). Naive Bayes and Text Classification I - Introduction and Theory. Recuperado desde <http://arxiv.org/abs/1410.5329>
- Saif, H., He, Y., Fernandez, M. & Alani, H. (2014). Semantic Patterns for Sentiment Analysis of Twitter. En *International Semantic Web Conference 2014* (pp. 324-340). Recuperado desde https://doi.org/10.1007/978-3-319-11915-1_21
- Schölkopf, B. & Smola, A. (2002). Support Vector Machines and Kernel Algorithms.
- Scott Gray, A. R. & Kingma, D. P. (2017). GPU Kernels for Block-Sparse Weights. Recuperado desde <https://s3-us-west-2.amazonaws.com/openai-assets/blocksparse/blocksparspaper.pdf>

- Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. En *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1631-1642).
- Thangaraj, M. & Sivakami, M. (2018). Text Classification Techniques: A Literature Review. *Interdisciplinary Journal of Information, Knowledge, and Management*, 13, 117-135. Recuperado desde <https://doi.org/10.28945/4066>
- Turney, P. D. (2002). Thumbs Up or Thumbs Down?: Semantic Orientation Applied to Unsupervised Classification of Reviews. En *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (pp. 417-424). Association for Computational Linguistics. Recuperado desde <https://doi.org/10.3115/1073083.1073153>
- van Atteveldt, W., Kleinnijenhuis, J., Ruigrok, N. & Schlobach, S. (2008). Good News or Bad News? Conducting Sentiment Analysis on Dutch Text to Distinguish Between Positive and Negative Relations. *Journal of Information Technology & Politics*, 5(1), 73-94. Recuperado desde <https://doi.org/10.1080/19331680802154145>
- Vinodhini, G. & Chandrasekaran, R. (2016). A Comparative Performance Evaluation of Neural Network Based Approach for Sentiment Classification of Online Reviews. *J. King Saud Univ. Comput. Inf. Sci.* 28, 2-12. Recuperado desde <https://doi.org/10.1016/j.jksuci.2014.03.024>
- Whitelaw, C., Garg, N. & Argamon, S. (2005). Using Appraisal Taxonomies for Sentiment Analysis. En *Proceedings of the 14th ACM International Conference on Information and Knowledge Management* (pp. 625-631). Recuperado desde <http://doi.acm.org/10.1145/1099554.1099714>
- Wirth, R. (1999). The CRISP-DM Process Model.
- Zhang, L., Wang, S. & Liu, B. (2018). Deep Learning for Sentiment Analysis : A Survey. *CoRR*, *abs/1801.07883*. Recuperado desde <http://arxiv.org/abs/1801.07883>
- Zhang, X., Zhao, J. J. & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. *CoRR*, *abs/1509.01626*. Recuperado desde <http://arxiv.org/abs/1509.01626>

Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H. & Xu, B. (2016). Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. *CoRR*, *abs/1611.06639*. Recuperado desde <http://arxiv.org/abs/1611.06639>

Zurada, J. M. (1992). *Introduction to artificial neural systems*. West publishing company St. Paul.

ANEXO A. RESUMEN EN INGLÉS

Abstract

Finding the sentiment or the polarity of a text has been one of the most researched tasks within the natural language processing field. Although initially the most important approaches were made through simple handmade analysis of the text, this has evolved into new techniques that take advantage of machine learning improvements for text classification. In the last years, the growth of the machine learning field has made significant advances compared with the traditional sentiment analysis methods.

This project's main purpose is to provide insight into the different machine learning techniques that can be applied to the text classification task, from baseline techniques to current state-of-the-art models. Apart from breaking down these models, they will also be made use of with a dataset from an important business rating service in order to determine which of the models performs better.

Having the opportunity to study the current available models is interesting, since nowadays, with the massive use of the Internet, more and more users use the social networks to give their opinions about the services or products they use. Thus, getting to know, in an automatic way, what users think, has become something really valuable.

Keywords: Machine learning; Sentiment analysis; Social networks; Artificial Intelligence; Word embeddings.

Introduction

The importance of social networks and their impact on society cannot be denied. According to recent statistics (Kemp, 2018), every month more than 3 billion people use social networks, which represents more than 42 % of the world's total population. From a research point of view, social networks are really useful as data sources, both for their large amount of available data and for the different possibilities they offer. This fact becomes interesting also for the business sector, being a great opportunity to gather valuable information about users and products through the use of data analysis and machine learning techniques.

In this context, we became aware of the Yelp social network, a site that allows people to post reviews about businesses and different services that openly provides its dataset for academic or research purposes. This dataset contains information about the businesses, the users and the reviews they post, including their images, which offers different techniques to apply, like data mining, graph mining or image recognition through deep learning among others.

Recently, the latest techniques in the text classification and sentiment analysis have been pushed forward, leading to new state of the art models with great results.

Motivated by these developments and the existence of the said dataset, we study some of the current techniques with theses objectives in mind:

- Understand the existing text classification machine learning approaches.
- Achieve results close to the ones from the state-of-the-art models.
- Serve as a general study of the different possibilities to do sentiment analysis.
- Carry out and research project in the field of artificial intelligence, applying the proper methodologies and reaching a final result.

State of the art

Sentiment analysis, also known as opinion mining, is one of the many tasks of the text classification field. Its main purpose is classifying the polarity of a text or opinion.

Traditionally, the sentiment analysis of a piece of text has been done through the application of several manual techniques (syntactic and semantic analysis). However, doing sentiment analysis this way is too costly, so researchers have been trying to find other methods that achieve good results but make it easier at the same time. This problem has been solved thanks to the improvements in the machine learning field.

Although machine learning has existed for a long time, one of the first times (and also one of the most important ones) it was applied to text classification was in Pang et al. (2002), where the **bag-of-words** model started gaining importance. The main problem when talking about text classification is that computers can't understand words as we humans do, so the focus has been on finding a way of converting human language to numbers so that computers understand text. What bag-of-words does, being the first model to represent words as numbers, is take the vocabulary of words in the text and build vectors based on the frequency of each term in the text. That way, terms that appear more often have a higher value, since they are the ones that carry more weight for the text polarity.

The bag-of-words model has been used along with the different machine learning algorithms, specially the supervised ones (Pang y Lee (2004) & Whitelaw et al. (2005)), so in this study we will try to get good results with some of the most relevant algorithms: Naïve Bayes, Logistic regression, Support Vector Machines and Random Forest.

While this model has resulted in significant results to solve sentiment analysis problems, its main disadvantage is that words end up being just numbers, instead of taking into account their meaning for context, which is important to predict the sentiment of a text. Consequently, researchers have made advances towards new

models that consider the context of the words, leading to the use of **word embeddings**. This kind of model was first introduced by Mikolov et al. (2013b) with the Word2Vec model. It consists on representing each word in a vector space so that there is a correlation between the word's position in the space and its meaning. Although Word2Vec was the first one, there have also appeared other powerful models, like GloVe and fastText, but we will be using Word2Vec as it has given good results.

Training word embeddings requires billions of words and is really time consuming, so the importance of these models has been boosted by the growth of the deep learning field. Using deep neural networks, pretrained word embedding vectors can be used by being “embedded” after the input layer, so the input text is multiplied by the pretrained weights. This way, convolutional networks and long short term memory networks have achieved great results when doing sentiment analysis, being the best results the ones by the models described in Johnson & Zhang (2017) and Johnson & Zhang (2016). This is the reason why we will be trying to replicate these models to solve our problem.

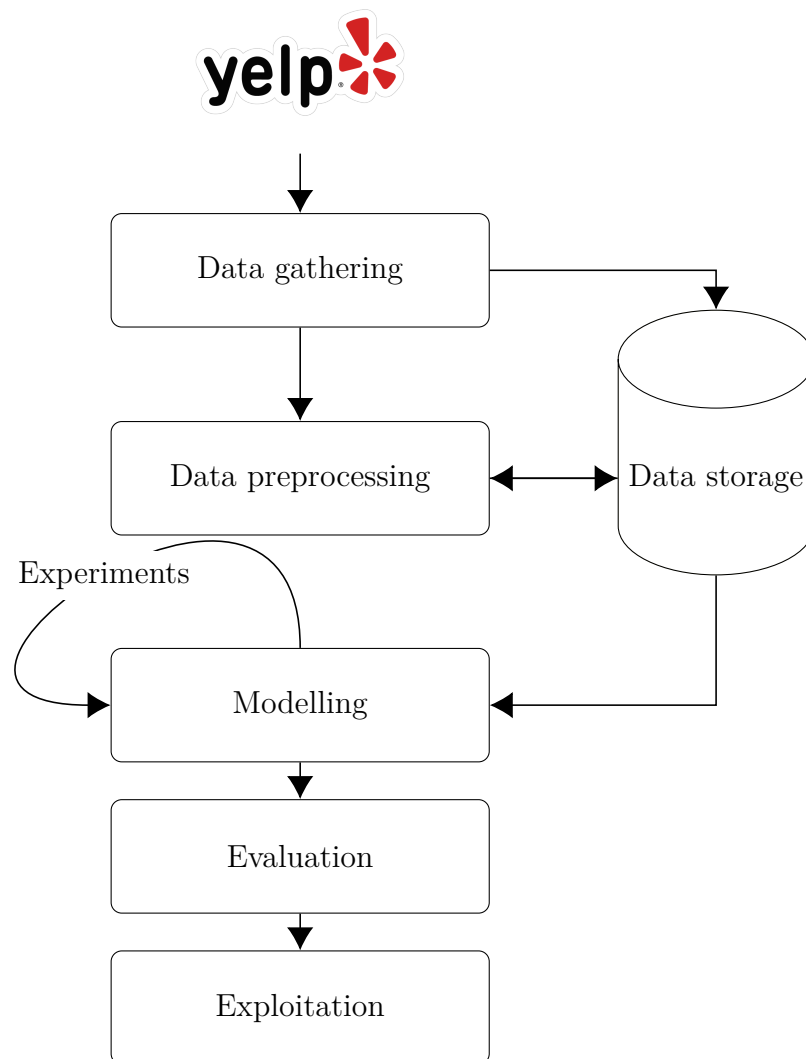
The use of word embeddings has highly improved the previous sentiment analysis and text classification models, but researchers have still tried to take things further. Therefore, they have focused on finding **universal word embedding models**, that is, word embedding models that work well for every text classification task, like sentiment analysis, machine translation, question answering and more.

In that context, very interesting models have shown up, like ELMo (Peters et al., 2018) and BERT (Devlin et al., 2018), but the best results have been provided by ULMFiT (Howard & Ruder, 2018), a model that employs an architecture with 3 long short term memory layers and that has reached 97,84 % accuracy on a sentiment analysis problem, the best result so far. As with the other types of models, we will try to apply this model to our problem and get good results.

Development

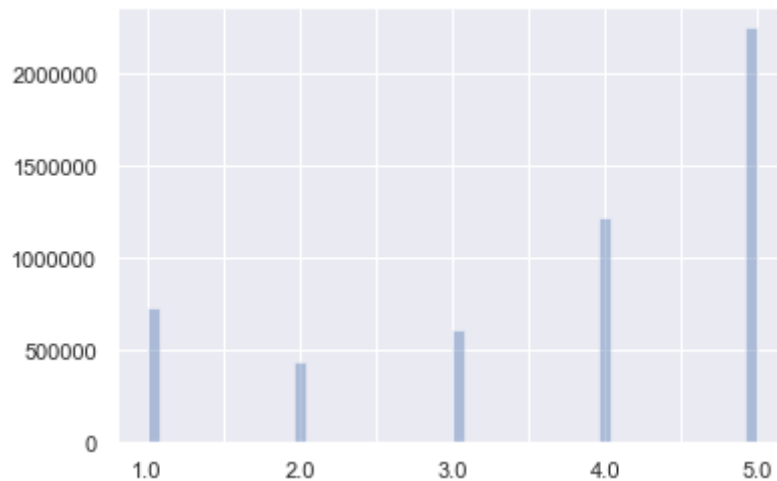
The problem this study tries to solve is to find a model that is able to predict the sentiment of a review from the Yelp dataset. The reviews in the Yelp website are rated with 1-5 stars, so our aim is, at the same time, to find a model that can predict the polarity of the reviews and to find a model that can predict the number of stars of a review. For this, the three types of models described in the state of the art section will be used.

When doing this kind of project, it is necessary to follow a structured methodology so that everything done is planned. Thus, this project has been approached through the next methodology:



The full dataset is obtained from the Yelp webpage, where they publish it freely. The dataset contains several .json files with information about users and businesses, but the only one we will work with is the review.json, where all the reviews and their rating can be found. There are a total of 5261669 reviews, but we will only work with a small part of them, since a machine learning model shouldn't need such an amount of data and it would be too time-consuming.

Once the data is obtained from the webpage, it is stored in a MySQL database. By observing the number of reviews for each star, it can be determined that there is a bias because the number of reviews is high for 5-stars reviews but pretty low for 1-star reviews. Thus, we normalize the data so that there is the same number of reviews for each star.



When building a machine learning model, one of the most important steps is pre-processing so that the model can handle the data better. But this step is even more important when working with text data, since text can be badly written. Therefore, necessary preprocessing steps have been done:

1. Normalization
2. Removal of repeated characters
3. Misspellings correction
4. Removal of punctuation and non-alphanumeric characters

5. Removal of stop words

Another preprocessing step that was going to take place was stemming the text, but a previous experiment that was carried out pointed out that stemming achieved worse results.

After all the reviews have been preprocessed, they are ready for the experimentation.

Experiments

In this section we report the experiments done with the available models already described.

Before beginning the experimentation and contemplating the setup, there has been a small previous experimentation with a simple setup to reach some conclusions to help us narrow down the different possibilities.

As stated previously, one option that was considered was to stem the reviews from the dataset, as another step to clean the text and have better input data for the models. Stemming reduces derived words to their word stem (root form), removing any suffixes. Performing this process could be of benefit for the models, since it reduces the size of the vocabulary (as different words with the same root form are contemplated as one) but, at the same time, suffixes provide meaning and context to the words, so removing them may cause the models to produce worse results. Furthermore, pretrained word embeddings have already been preprocessed in a different way than our text, so stemming it might not be beneficial.

Input dataset	Model					
	Naïve Bayes		CNN+LSTM		ULMFiT	
	Binary	Multi.	Binary	Multi.	Binary	Multi.
Clean data	80,43 %	51,33 %	85,89 %	58,94 %	85,25 %	55,61 %
Stemmed data	80,01 %	50,94 %	83,39 %	55,68 %	84,30 %	54,55 %

The results clearly show that stemming the data performs worse in all cases but the results that stands out are the ones with the word embeddings. Having looked into this fact, we try to see the impact of using a bigger dataset.

Initially, we planned to start the experimentation with a dataset consisting of 125000 reviews, but taking more reviews as input to the models may mean a great outperformance, so we make experiments with the mentioned dataset and another with double the amount of reviews to see how it affects the output.

Input dataset	Model					
	Naïve Bayes		CNN+LSTM		ULMFiT	
	Binary	Multi.	Binary	Multi.	Binary	Multi.
125000 reviews	80,43 %	51,33 %	85,89 %	58,94 %	85,25 %	55,61 %
250000 reviews	81,06 %	52,20 %	86,05 %	59,39 %	85,56 %	56,21 %

From the outcome it can be concluded that, obviously, using a bigger dataset results in better results. Given that, using double the reviews also doubles the training time for the models, but the performance of the bigger dataset is not such as to consider using it.

On the explained basis, with the 125000 cleaned reviews dataset, we will follow a training protocol using cross validation to evaluate the performance of the parameter tuning. Thus, the data will be divided into 5 folds and each fold will be a training round where the data will be splitted into training and validation sets different from the other folds.

The evaluation metrics used to validate the models and choose between parameter configurations will be accuracy, f1-score, ROC-AUC and the cross-entropy loss.

For the bag-of-words model, the Naïve Bayes, SVM, Random forest and Logistic regression algorithms will be used. Each of these algorithms have their own parameters to tune, so, for each of them, there is a set of predefined parameters on the basis of which an exhaustive search is carried out and then, if any trend is perceived, further experimentation is done.

In the case of the word embedding models, two types of neural networks are trained: one with the DPCNN architecture from Johnson y Zhang (2017) consisting in multiple blocks of convolution and pooling layers, and another with the BLSTM model from (Zhou et al., 2016) that has a bidirectional LSTM layer followed by a convolutional one. The training done with both architectures is the common neural network training process, altering the number of neurons, the learning rate applied to the net, the optimizer and the number of epochs according to an early stopping regularization and watching how these changes impact the performance of the

model.

The ULMFiT model from Howard y Ruder (2018) is, actually, a model of neural network (inspired by Merity et al. (2017)) called AWD-LSTM that uses LSTM layers so, in fact, the training done is similar to the described for the word embedding nets. However, it has an added intricacy and it is that the training is done first for the pretrained model in a fine-tuning process and then the normal classifier is trained with discriminative fine tuning and gradual unfreezing of the net's layers.

The set of parameters to be tested with the bag-of-words model can be found in the following table:

Parameters	Values
Vector transform	Ocurrencias Frecuencias inversas
Ngrams	Unigrams Unigrams and bigrams
Sublinear frequency	Yes No
Naïve Bayes	
Alpha	1 0,5 0,1 0,01 0,001
SVM	
C	0,01 0,1 1 10 50
Random Forest	
Number of trees	50 100 200
Logistic regression	
C	0,001 0,1 1 10 100 500

While the bag-of-words model will be tried out by using the set of parameters and further experiments will be done incase any special trend is found, the word embedding nets training process is manual, trying different configurations and carrying on with the ones that provide better results. Particularly, the training focus on trying the hyperparameters that the authors of the models have proved to work well: the Adam optimizer and a learning rate of 0.001 with an l2 regularization of 0,00001 for the DPCNN architecture and a learning rate of 1,0 regularized in the same way with 0,5 dropout to the embeddings layer, 0,2 for the bidirectional layer and 0,4 for the convolutional layer in the case of the BLSTM net. ULMFiT is trained through a

library created by the author that contains the model implementation and its best parameters: 0,4 dropout applied to the embedding layer, 0,3 for the LSTM layers and the Adam optimizer with a learning rate of 0,004 for fine tuning and 0,01 for training.

Results

Below are represented the sets of results obtained via the previously described experimentation. As stated before, the evaluation metrics used are the accuracy, the f1-score and the area under the ROC curve, also focusing on minimizing the cross-entropy loss.

The results achieved by the bag-of-words model are shown in the following tables (one for each dataset, binary and multiclass).

	Accuracy	F1	ROC-AUC	Cross-entropy loss
Naïve Bayes	83,99	83,61	0,9194	0,1675
SVM	87,22	87,19	0,9452	0,3008
Random Forest	82,92	82,92	0,9120	0,4828
Logistic regression	87,19	87,16	0,9450	0,3489

	Accuracy	F1	ROC-AUC	Cross-entropy loss
Naïve Bayes	55,30	55,57	0,8053	1,0767
SVM	59,65	59,25	0,8652	0,9658
Random Forest	52,40	50,85	0,7610	1,2816
Logistic regression	59,25	58,92	0,8605	0,9619

Overall, the best results of the model have been provided by using unigrams and bi-grams for the construction of the vocabulary as, cosequently, the models understands better the context of the words, sublinear frequency and vectorization through inverse frequency, the last one because it penalizes words that appear to much but don't have an important impact. Nonetheless, the Random Forest algorithm has worked out the other way round, since the use of unigrams have provided best results.

As it can be seen, SVM and logistic regression have a better performance. This might be due to the fact that the other ones, Random Forest and Naïve Bayes, are classifiers that are based on logic and probability, respectively, while the first ones are based on analytic functions that make linear combinations of the characteristics, being better at approaching the feature values of the reviews.

Moving on to compare these results with the ones acquired by the models that use word embeddings, the next table collects the results of these models and compares them with the best result of the bag-of-words model.

	Accuracy	F1	ROC-AUC	Loss
SVM (bag-of-words)	87,22	87,19	0,9452	0,3008
DPCNN (word embed.)	87,35	87,06	0,9477	0,3141
BLSTM (word embed.)	88,00	88,00	0,9514	0,2924
ULMFiT	89,72	89,52	0,9647	0,2680

	Accuracy	F1	ROC-AUC	Loss
SVM (bag-of-words)	59,65	59,25	0,8652	0,9658
DPCNN (word embed.)	61,03	60,49	0,8811	0,9368
BLSTM (word embed.)	61,68	60,25	0,8704	0,9025
ULMFiT	63,67	62,98	0,8859	0,8753

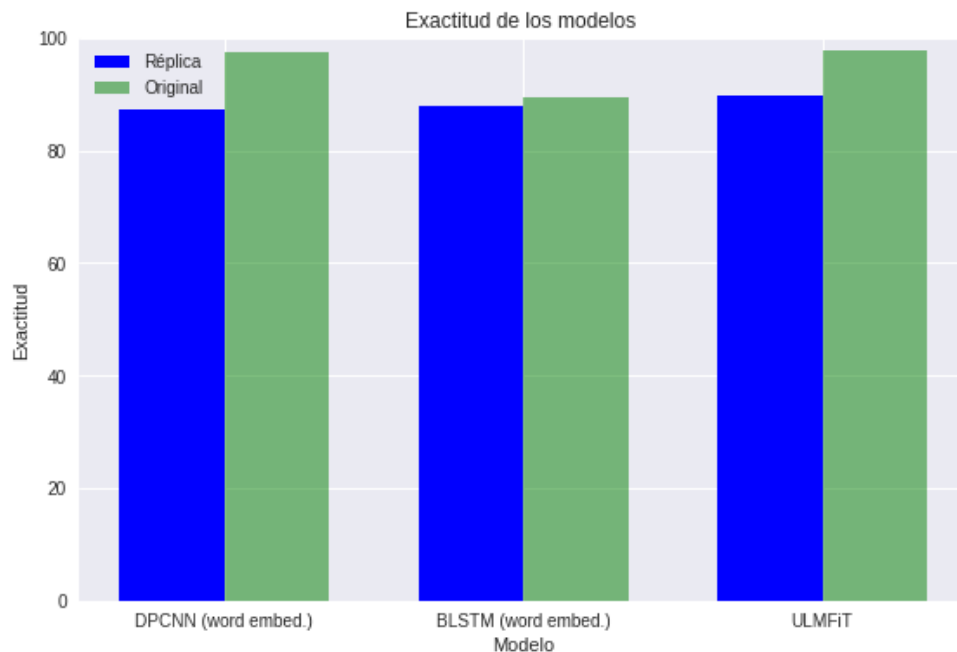
In general, great results have been achieved in both datasets. In the case of the binary one, since it only has two categories/polarities to classify, great models have been built that get close to a 90 % of accuracy. Classifying five different categories (the stars) is much more difficult because there isn't a great difference between the middle stars (2, 3 and 4), which makes it harder for the model. Even so, the models have got good results, around 60 % of accuracy, taking into account that simply throwing a dice would lead to a 20 % chance of choosing the right star.

The best model has ended up being ULMFiT in both datasets. We have managed to get a final model that classifies the polarity of the reviews with an accuracy of 89,72 %, which is a great result, since it isn't far from the state of the art results

and increases in 2,5 % the best model obtained with the bag-of-words model and in 1,72 % the best model using the nets with the pretrained word embeddings. For the multiclass dataset, the same happens, although in this case ULMFiT performs even better than the other models, since the difference with the SVM is of a 4,02 % of accuracy. The final models draw positive conclusions, as they manage to classify 5 stars with up to a 63,67 % of accuracy and, although it could seem pretty lower than the results obtained with the binary dataset, getting a model that correctly classifies 5 different categories is harder, which has been seen in section 2.5, as state of the art models still try to solve this problem, with ULMFiT reaching the best result with 70 % accuracy.

Particularly between the word embedding nets, the one with the best result is the one that uses LSTM neurons, the same type of neurones used by the best state of the art model (ULMFiT), although at first it was expected of the DPCNN model by Johnson y Zhang (2017) to get better results according to the ones reported by the authors.

It can provide better insight to compare the obtained results with those of the original models that we tried to replicate (seen in the table 2.2).



Considering the plot, although DPCNN got great results, close to those of ULMFiT,

we didn't manage to get really close to them, having a difference of 10,01 % of accuracy in contrast to our model. However, our BLSTM model is only 1,5 % away from the original one, getting a similar result, although it's true that the original results of BLSTM were way lower than the DPCNN ones.

As expected, ULMFiT, which was the model with best results in the state of the art, is also the model which provided the best results to solve our problem, although there was still a gap of 8,12 % compared to the results posed by the authors.

Beyond the accuracy results, the models can be said to have achieved great results. Before the experimentation, it could have been thought that the machine learning algorithms with the bag-of-words model were going to perform worse than the others, but they ended up getting results pretty similar to the ones got by the nets with word embeddings, even though the original models actually do greatly outperform the bag-of-words models.

When trying to solve the reviews classification problem, the exploitation would be done with the ULMFiT model, since it is the one with better results, but the bag-of-words model should be taken into consideration, since it also gets good results and its training complexity and time is way lower.

Conclusion

The work done has solved the problem of social network reviews classification through the usage of different types of machine learning models. The state of the art possibilities have been studied and applied, including machine learning algorithms with the bag-of-words model, neural networks with word embeddings and the ULMFiT model, which is one of the new type of models that try to universalize the use of trained word embeddings for multiple tasks.

With them, different models capable of classifying reviews in both positive/negative and 5 stars have been accomplished with results close to the ones of the state of the art, with a 89,72 % of accuracy for the first type and 63,67 % for the second one. This way, not only the classification problem has been solved but we have managed to offer an analysis and comparison between each type of model.

Regarding the project as a whole, a research work was carried out, studying all the state of the art in the first place, analysing it to see how it can be applied to the problem and doing an experimentation according to the methods and techniques learned through the degree.